MICROPROGRAMME DEVELOPMENT FOR A BIT - SLICE SYSTEM USING SIMULATION

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

by

SHEKHAR KUMAR GHOSH

to the

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR
FEBRUARY, 1986

Th
001.6423
G.346 m

DEDICATED TO

My Parents

CERTIFICATE

This is to certify that this project entitled, 'MICROPROGRAMME DEVELOPMENT FOR A BIT-SLICE SYSTEM USING SIMULATION' by Sekhar Kumar Ghosh has been carried out under my supervision and that it has not been submitted elsewhere for the award of a degree.

[illegible]

February, 1986

R.N. Biswas
Professor
Department of Electrical Engineering
Indian Institute of Technology
Kanpur.

ACKNOWLEDGEMENTS

I take this opportunity to express my sincerest thanks and deep sense of gratitude to my thesis supervisor, Dr. R.N. Biswas, without whose help and guidance this work would not have taken shape. His instructions and suggestions have proved to be of immense help at every stage of progress.

I am very grateful to M/s. Hindustan Aeronautics Limited, Hyderabad and Korwa Division for giving me an opportunity to pursue the M.Tech. course at I.I.T. Kanpur.

A special note of deepest appreciation is extended to my wife, Alpana, my son, Deep and other friends who have contributed largely in making my stay at I.I.T. Kanpur pleasant and memorable.

Last, but not the least, I must thank Mr. C.M. Abraham for doing an excellent job while typing the manuscript.

- Sekhar Kumar Ghosh

ABSTRACT

An attempt has been made to develop a development system for bit-slice based microcomputers. The microinstruction structure in such a system strongly depends on the system architecture. An architecture for a 16-bit microcomputer system has, therefore, been evolved with four Am 2903 Bit Slice Processor, one Am 2902 Carry - Look ahead Generator, one Am 2914 Priority Interrupt Controller, one Am 2910 Microprogramme Controller and appropriate registers, counters and multiplexers with special emphasis on the effective use of a 64-bit microinstruction structure. To judge the effectiveness of this structure, microprogrammes have been written for executing the instructions of one standard microprocessor the 8085 A. A scheme has been suggested to build a Simulator for such a system so that the microprogrammes and the interaction with peripherals can be tested without assembling the actual hardware. Actual PASCAL programmes for simulating a 16-bit processor module and an Am 2910 Microprogramme Controller have been written and verified on an INTEL Series III Microcomputer Development System.

Page

Contents

CHAPTER 1

INTRODUCTION

LSI, VLSI technology leads to a revolution in the Computer Industry. Due to the development of modern sophisticated process technology, it has become possible to manufacture LSI and VLSI IC chips within reasonable cost. This has made it possible for the manufacturing industry to build computers with considerably low cost, resulting in the proliferation of the applications of computers in many fields. Due to such wide applications, the requirement of the computing system has also changed. It is no longer possible, or even advisable, to use a specific type of computing system to cater for different type of needs. Design and architecture of the computing system has changed to satisfy various user for their specific needs.

Design of computers is now highly systematic and modular process, enabling the designer to evolve specific architecture of the computing system to meet the specific needs of the user.

Bit-slice microprocessors are very powerful logical building blocks for such modular designs. By virtue of its modularity and inherent high speed, it is widely used in system like CPU's, peripheral controllers, programmable microprocessors,CRT controllers etc. However, the design of bit-slice based computing system is rendered quite difficult and tiredsome, because, complex microprogramme needs. A simulator for a bit-slice processor based system will be a powerful and helpful design tool.

A simulator is a development system which helps and guides the designer to realize his ideas into practice, without actually assembling the hardware. In the initial stage of development, one can avoid using chips, one can practice or implement micro-programming and cross check if it is working properly. A simulator with built-in diagnostic programme can guide and help the designer to avoid errors (like bus conflicts, micro-control errors, I/O port clashes and hardware errors, etc.). It reduces the cost of development and builds confidence among designers to make a complete and a suitable computing system.

A useful tool for the development of microprogrammed systems is the software Simulator, a programme written to simulate the precise behaviour of the data flow from the point of view of the microprogramme. The programme simulates the underlying hardware design by executing the microprogramme. Hence, the Simulator allows one to test and debug the microprogramme before the hardware system is available.

The advantages of such Simulators are the following :

- They allow one to test, debug, and optimize the micro-programme before the hardware is available.

- They give one a more flexible and convenient vehicle for microprogramme testing.
- They can contain debugging and instrumentation functions.
- They can contain checks for error situations, such as the detection of timing errors and bus conflicts.
- By allowing the testing of the microprogramme to begin earlier, the use of the simulator can provide valuable feedback on the hardware design, both in terms of errors and possible optimizations.

This project has been carried out with the following objectives:

1. To evolve an architecture of a Bit-Slice based microcomputing system.
2. To specify and structure microinstructions format for the architecture mentioned in Step 1.
3. To simulate LSIs and other functional blocks used in the computing system and combine them to obtain a complete computing system.
4. To write microprogrammes, using the choosen microinstruction format, for a suitable set of instructions so as to enable the user to write a programme in terms of the instruction set to obtain the corresponding object codes.

Chapter 2 is an appraisal of the various design alternatives culminating in the justification and description of the proposed system design. The microinstruction format has been evolved in

Chapter 3, followed by a discussion on the implementation of popular Intel 8085A instruction set. By using four of the Am 2903 (4-bit slice) processor, a 16-bit module has been simulated, and it is described in Chapter 4. The development of the Am 2910 Simulator is stated in Chapter 5. Conclusion is discussed in Chapter 6.

CHAPTER 2

CPU SYSTEM CONFIGURATION

The proposed system configuration is shown in Fig. 2.1. It is a 16-bit machine. It consists of a 16-bit Arithmetic Logical Operation Management Unit (AOMU), connected to one system Data Bus (DB) and one system Address Bus (AB).

The Microprogramme Management Unit (MMU), schedules the next microinstruction to be executed by the system. It is assumed that the main programme will be residing in the main Memory and each instruction (Macro), when fetched from the main memory, is brought onto the Data Bus and then decoded by the MMU - resulting in a predetermined sequence of micro-instructions being executed.

The Interrupt Management Unit (IMU), provides a well defined way of altering the flow of status in response to outside asynchronous events.

2.1 ARITHMETIC/LOGICAL OPERATION MANAGEMENT UNIT (AOMU)

The AOMU contains the following logical building blocks as shown in Fig. 2.2.

1. The RALU is a 16-bit parallel subsystem consisting of four, 4-bit wide Am 2903 bit-slice processors alongwith a high-speed Carry-Look ahead generator (Am 2902).

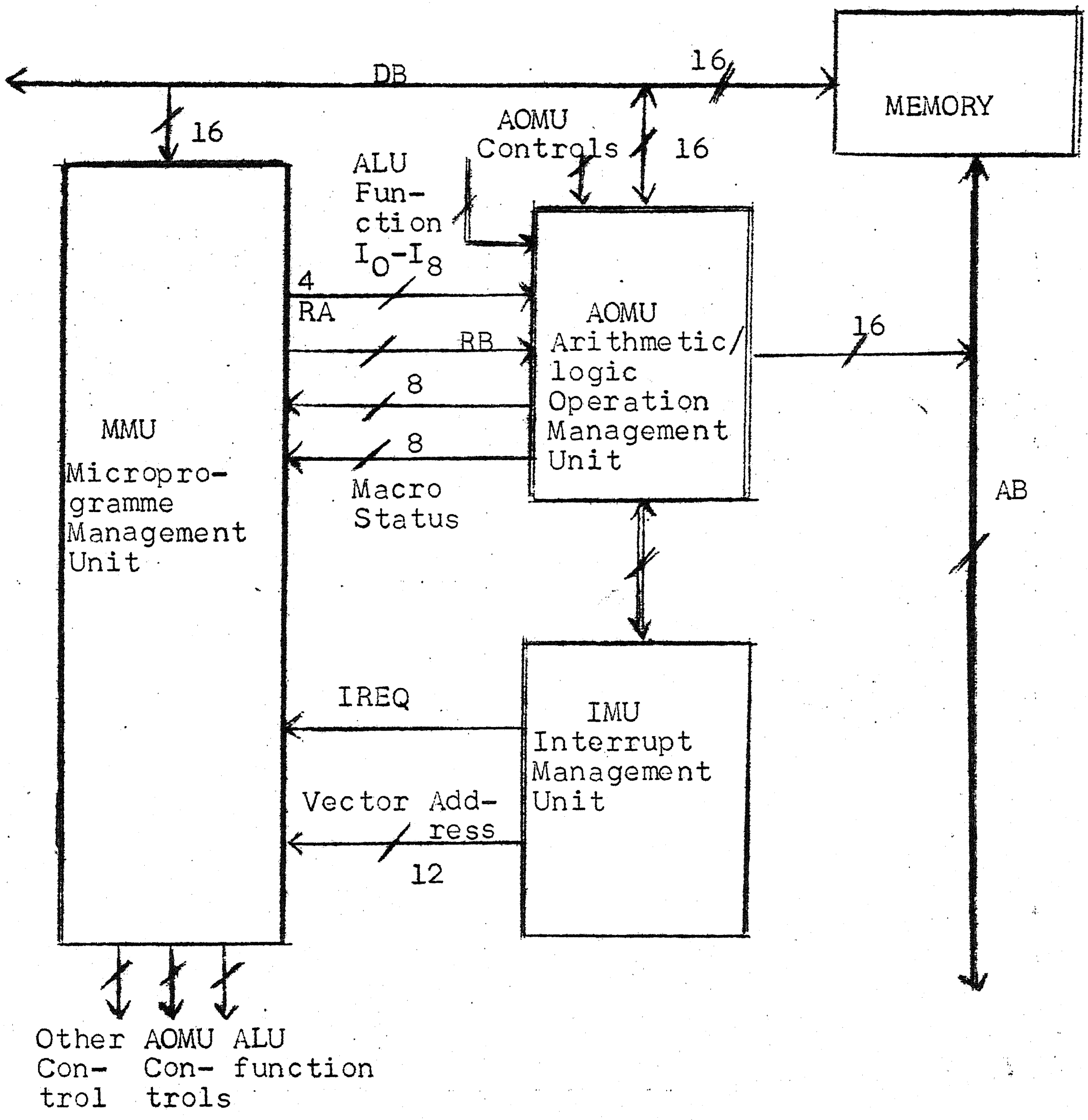


Fig. 2.1 CPU Block Diagram

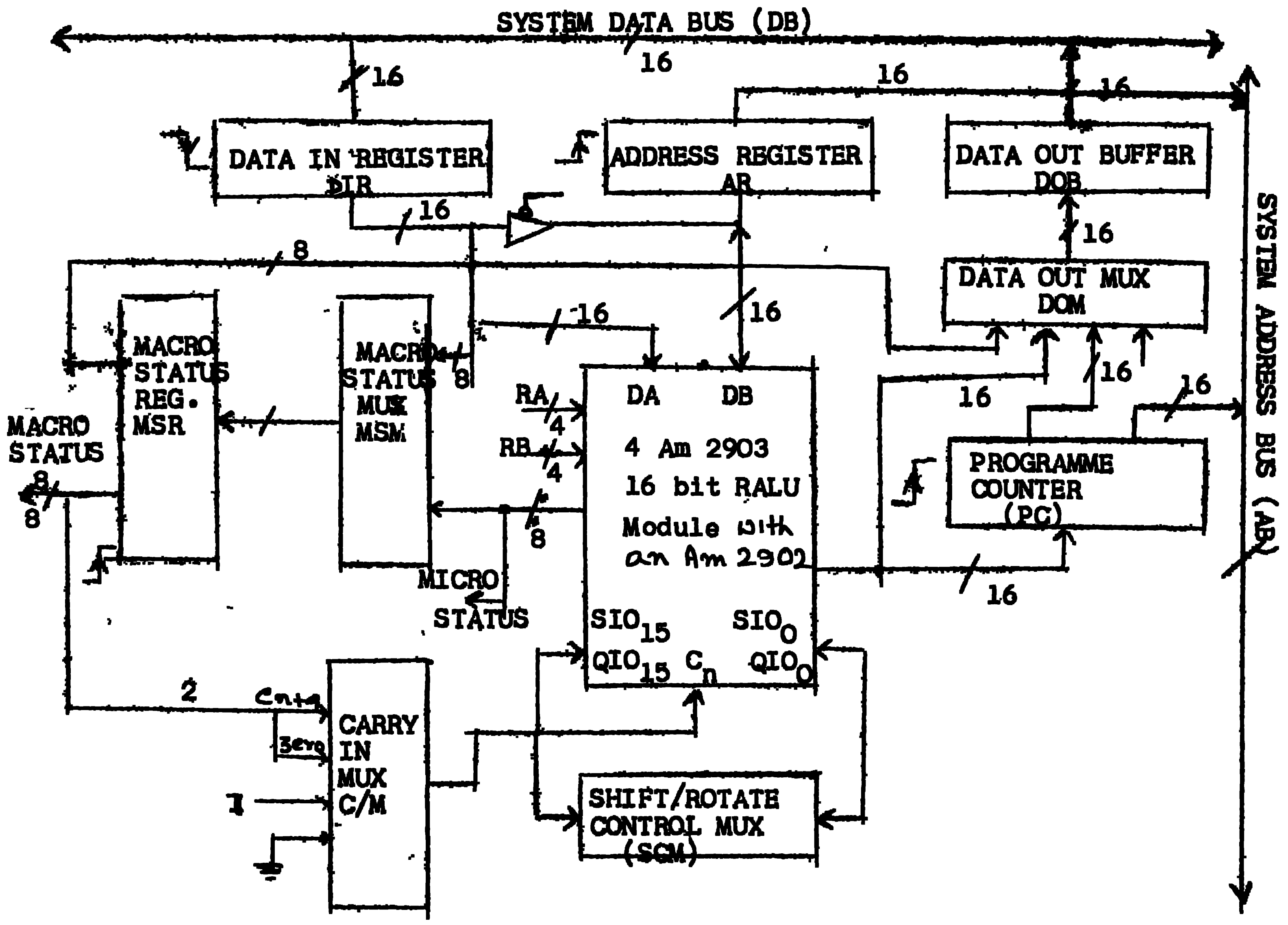


Fig. 2.2 ALU Operation Management Unit (AOMU)

2. Data-In Register (DIR).
3. Address Register (AR).
4. Macro Status Register and Macro Status Multiplexer (MSM).
5. Carry-In Multiplexer (CIM).
6. Shift/Rotate Control Multiplexers (SCM).
7. Programme Counter (PC).
8. Data-Out Buffer (DOB).
9. Data Output Multiplexer (DOM).

2.2 MICROPROGRAMME MANAGEMENT UNIT (MMU)

Fig. 2.3 shows the configuration of the MMU. It consists of the following logical building blocks.

1. A 16-bit Instruction Register (IR).
2. One Mapping PROM (MP).
3. One Microprogramme Controller (MC).
4. Microprogramme Memory (MM).
5. Pipeline Register (PR).
6. Condition Code Multiplexer (CCM).

The Microprogramme Controller uses the Pipeline Register and allows parallelism during the execution of a microinstruction. Its main purpose is to split the total delay of the system into two delay paths. One path runs from the Microprogramme Contro-ller, through the Microprogramme Memory, and into the Pipeline Register. The second path runs through the CPU into the Status Register. Since these two paths are active simultaneously, the

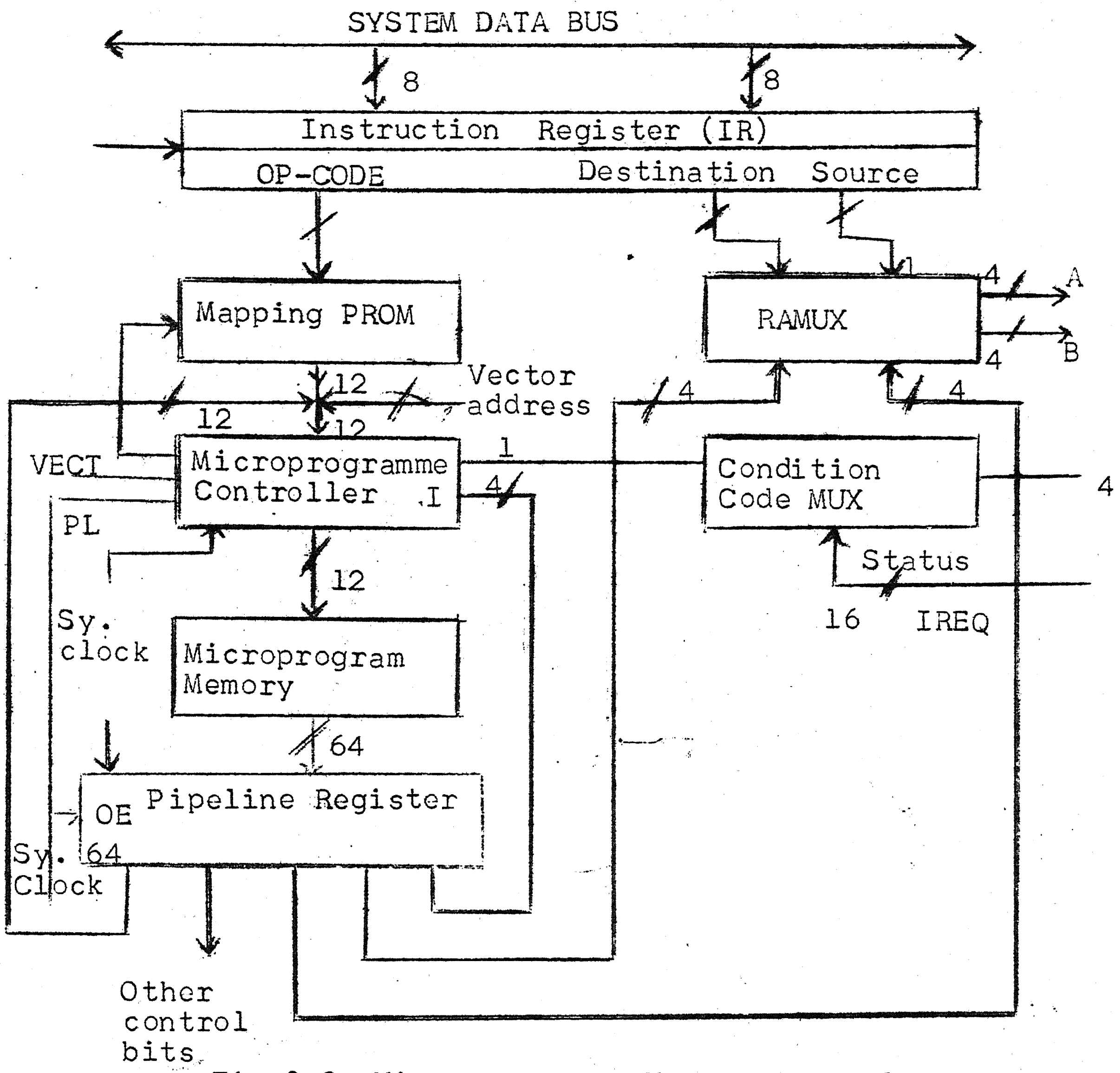


Fig.2.3 Microprogramme Management Unit

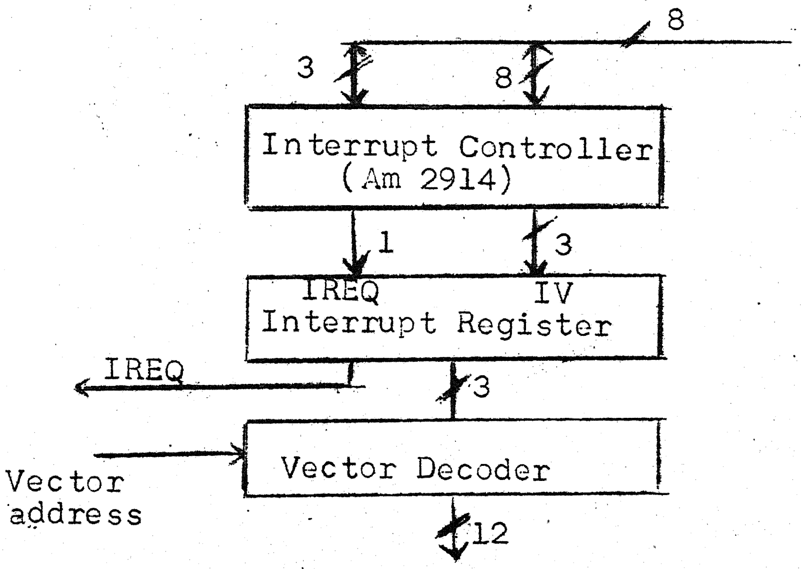


Fig. 2.4 Interrupt Management Unit

machine cycle time is determined by the delay of the longer of the two paths. The Pipeline Register contains the microinstruction currently being executed. The data manipulation control bits go out to the system elements and the next address field of the microinstruction is returned to the sequencer to determine the address of the next microinstruction to be executed. The address is sent to the Microprogramme Memory and the next microinstruction is available at the input of the Pipeline Register. The presence of the Pipeline Register allows the microinstruction fetch to occur in parallel with the data processing operations.

2.3 INTERRUPT MANAGEMENT UNIT (IMU)

The IMU contains the following logical building blocks, as shown in Fig. 2.4.

1. A Priority Interrupt Controller.
2. Interrupt Register.
3. Vector Decoder.

2.4 OP-CODE EXECUTION

During a 'FETCH' request to the host machine, the op-code, floated on the Data Bus from the main system memory, is clocked onto the Instruction Register. The most significant 8-bits of the Op-code decode a 12-bit address (corresponding to the starting address of the op-code) and go as direct inputs to the Microprogramme Controller and are selected as the

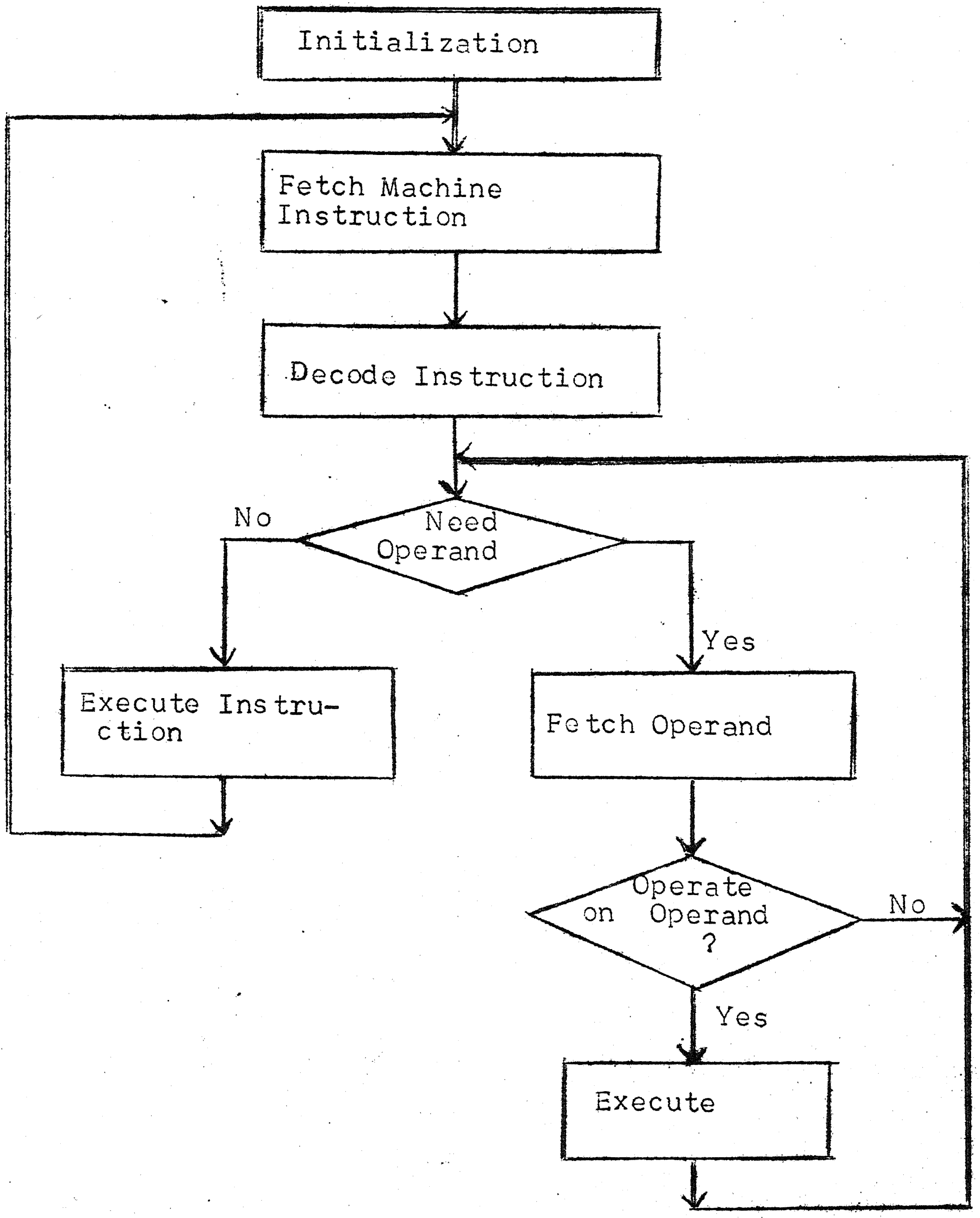


Fig. 2.5 OP-Code Fetch Flow Chart

starting address of the corresponding microprogramme to be executed. The output of the Microprogramme Memory (the microinstruction) is clocked onto the Pipeline Register, which sends controls to the various functional blocks of the system. A microprogramme for the execution of an OP-code is a sequence of one or more such instructions. The last micro-instruction again activates the fetch cycle for the next OP-code. A flow chart of OP-code fetch execution is shown in Fig. 2.5.

2.5 RALU

The Am 2903 is a high-performance RALU capable of performing seven arithmetic and nine logic operations. It can also perform nine special functions on two four-bit operands. The control signals $\overline{E}_A$, $\overline{OE}_B$ and I_o decide the RALU operand sources. Operands can be choosen from internal RAM output A, RAM output B, from external sources like DA, DB and from internal Q register content. The control signals I_1-I_4 decide the RALU functions, whereas I_5-I_8 determine the destination, shift and rotate functions. When I_0-I_4 control signals are low, the Am 2903, executes special functions. When $\overline{I}_{EN}$ is low, it enables writing onto the Q register and pulls the $\overline{WRITE}$ output low. Since $\overline{WRITE}$ will be connected to $\overline{WE}$ input of local RAM, it will enable writing into the local RAM addressed by B. The Y output buffers are enabled when the $\overline{OE}_Y$

control signal is low and are in high impedance state when $\overline{OE}_Y$ is high. (For details, refer to AMD Data Manual).

2.6 DATA IN REGISTER (DIR)

Data In Register is a 16-bit negative edge triggered register. When DIR input is enabled with a control signal, data from the Data Bus can be taken onto the register and can be passed onto the DA input of Am 2903. The content of the DIR also can be passed onto the DB bidirectional bus of Am 2903 under the control of $\overline{OE}_B$ signal.

By using DIR, data can be taken onto the Am 2903 from the system bus. This is the only path through which data can be taken onto the Am 2903. By using DB as another data input port, the same data from the DIR can be brought to S operand of the Am 2903. Any ALU operation can be achieved on the same data by keeping it in R and S operand sources. In a system where dual Data Bus or Address Bus structure is existed - the DB can be connected to one of those two buses. In some configuration ALU operations are performed on a set of predetermined data. Usually predetermined data will be stored into a PROM. The DB bus can be connected to the PROM output. Since in this system ALU operations are not done on any predetermined data, neither we have dual data bus structure, DB is connected to DIR only through a tri-state control.

2.7 ADDRESS REGISTER (AR)

This is a 16-bit, positive edge triggered register. The operands addresses for different addressing modes can be calculated in the ALU, and these addresses can be loaded onto th Address Bus through its output enable control. This is one route to access the Address Bus.

2.8 MACRO STATUS REGISTER AND MACRO STATUS MULTIPLEXER (MSM)

Each time the ALU is used, it generates four main status (carry out, overflow, zero, and sign). Apart from these status, it also generates four other status on SIO_n, SIO_0 , QIO_n, QIO_0, bidirectional I/O lines (like parity, etc.). Thes status outputs, as well as previous status word (PSW) can be stored in the Macro Status Register, if its input is enabled. Previous status word can be obtained from the main memory through the DIR and MSM, which requires one control signal to select the status word to be stored out of PSW and the present macro status from the RALU.

In this configuration (MSM and Status Register), one can test micro status and macro status in the Condition Code Multiplexer.Macro status may be used in conditional macro jump, whereas micro status help in conditional micro jump.

2.9 CARRY-IN MULTIPLEXER (CIM)

In an ALU operation the least significant slice (LSS) requires a carry-input signal (C_n). In different conditions,

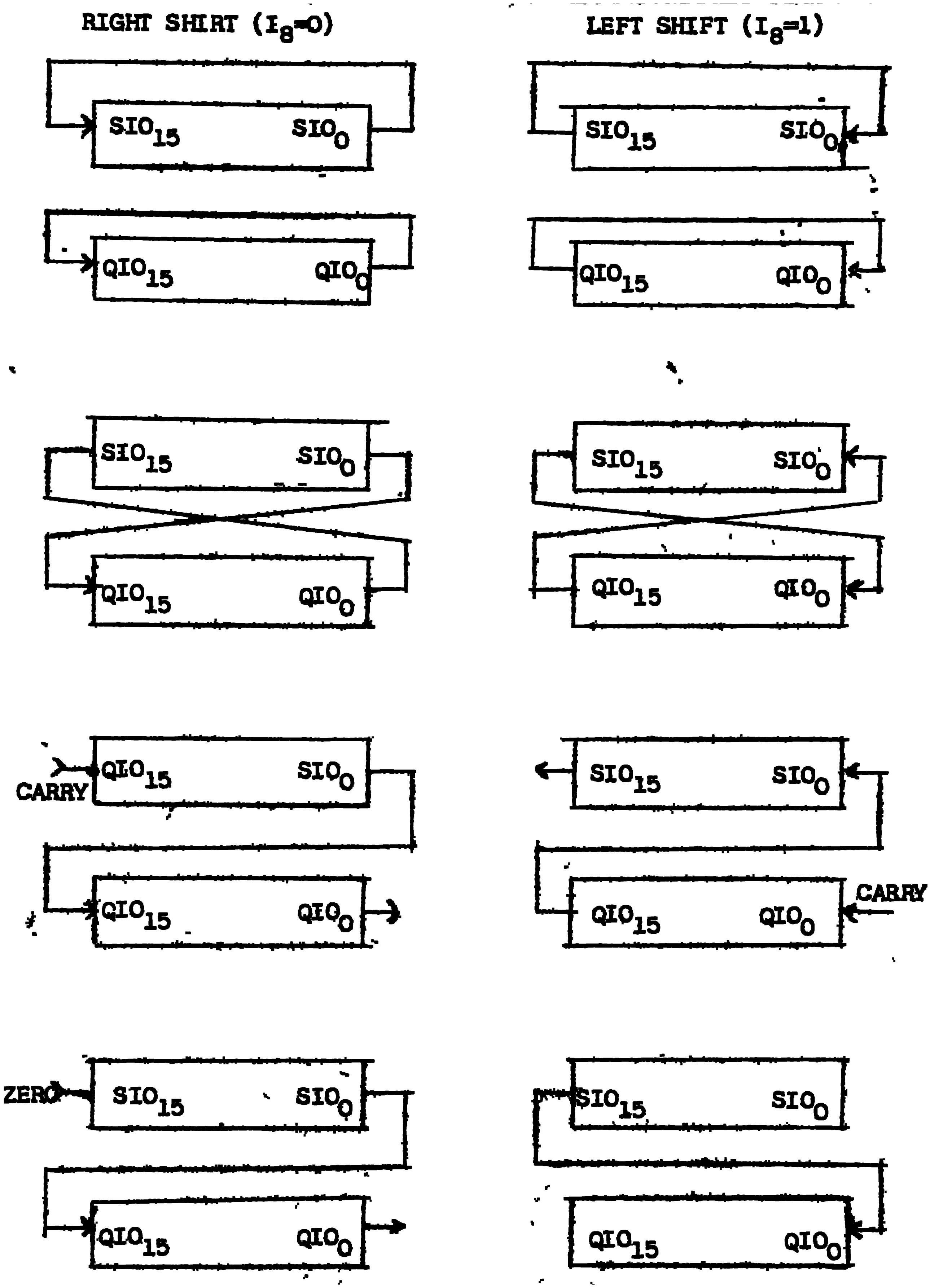


Fig. 2.6 Shift and Rotate Linkages

this signal can have a value from 0,1, carryout (previous carry from most significant slice) or zero (previous status). A Carry in Multiplexer (CIM) is used to select the required signal. Two control signals are required to select the required carry input value onto the ALU.

2.10 SHIFT ROTATE CONTROL MULTIPLEXERS (SCM)

In Am 2903, ALU output (F) can be shifted up, and down. For shift and rotate operation SIO_n, SIO_0, QIO_n, QIO_0 bi-directional I/O lines are used. Am 2903 can perform for long shift, short-shift and rotate operations. In this system a wide range of choice is provided for shift and rotate inputs. By virtue of all these inputs, it is possible to execute various type of shift and rotate operations, offered by various instruction sets. The ALU control signal I_8 controls the tri-state outputs of the multiplexers. ($I_8 = 0$ implies right shift and $I_8 = 1$ implies left shift operation). It requires three control signals to select the required input for shift and rotate operation.

The potential shift and rotate linkage are shown in Fig. 2.6.

2.11 PROGRAMME COUNTER (PC)

It is a 16-bit synchronous counter which keeps track of the addresses of the microinstructions in the main memory when instructions are fetched from the main memory. The increment and loading operations are done at the positive edge of the clock. The PC can be loaded only from the Am 2903 output (Y). Under a control signal the PC (address) can be enabled onto the system address bus. This is the second route to access the address bus. Two control signals (for loading the address, and for incrementing the counter content) are required to operate PC.

2.12 DATA OUT BUFFER (DOB) AND DATA OUTPUT MULTIPLEXER (DOM)

When enabled by a control signal, data from the AMU can be passed on the data bus. Data from three possible sub blocks can be loaded onto the data bus, through the data output multiplexer. They are - macro status, PC address, and the ALU output (Y). The data from the data bus can be loaded to the main memory, when it is addressed by the AR and memory write signal is enabled. Instead of using three different buffers and few extra control signals, only three control signals are used (two for DOM, and one for DOB), to achieve the same function.

2.13 PRIORITY INTERRUPT CONTROLLER, INTERRUPT REGISTER AND VECTOR DECODER

To handle interrupts into this system, an Priority Interrupt Controller (PIC) (Am 2914), an Interrupt Register and a Vector Decoder are used. The Am 2914, is a high-speed, 8-bit priority interrupt unit, that is cascadable to handle any number of priority interrupt request levels. It can receive interrupt requests on 8-interrupt input lines. An 8-bit mask register is used to mask individual interrupts. An internal status register is used to point to the lowest priority at which an interrupt will be accepted. These mask and status registers can be loaded from the Am 2903 output (Y). The Am 2914 is controlled by a 4-bit instruction control field I_0-I_3. The command on the instruction lines is executed if IE (instruction enable) is low and is ignored if IE is high, allowing the 4-I bits to be shared with other devices. Upon receiving interrupt requests from asynchronous peripherals, the priority interrupt controller checks the priorities of the input interrupts and compare with the mask register value. A higher priority interrupt and a Interrupt Request signal will be passed onto the Interrupt Register as a Interrupt Vector signal and IREQ signal. The interrupt register will store the interrupt vector signal as well as IREQ signal. The IREQ signal will be passed onto the Condition Code Multiplexer - so that the Microprogramme Controller (MC) can test this status condition, whenever it is required.Vector Decoder receives, interrupt vector signal and decodes to a 12-bit address, which will be connected to the D-input of the MC, which indicates the MC, the starting address of the interrupt service routine.. Micro and macro type of interrupts can be handled in this configuration.

2.14 16-BIT INSTRUCTION REGISTER (IR)

The most significant 8-bits correspond to the OP-code followed by 8-bits for the 'A' and 'B' addresses for the scratchpad registers of the RALU (Am 2903). 0 to 3 bits of IR denote the source and 4 to 7 bits denote destination address of the operands in the RALU. The clock to the IR can be enabled by a control signal and when clock goes high to low, it latches the DB contents to IR. A macro instruction from the main memory can be enabled to IR by enabling its input.

2.15 REGISTER ADDRESS MULTIPLEXER (RAMUX)

It is used to select the source and destination registers (address), either from IR or from the microinstruction control bits (PL) in the pipeline register outputs.

2.16 MAPPING PROM (MP)

It is a 8x12 bit PROM, stores the starting address of each OP-code. When it is addressed from IR, it generates a 12-bit starting address, corresponding to a OP-code. If its output is enabled by $\overline{\text{MAP}}$ control signal, this 12-bit address is passed on to the D-input of the Microprogramme Controller.

2.17 MICROPROGRAMME CONTROLLER (MC)

The Am 2910 Microprogramme Controller is an address sequencer, intended for controlling the sequence of execution of microinstructions stored in a Microprogramme Memory. Besides the capability of sequential access, it provides conditional branching to any microinstruction within its 4096 microword range. During each microinstruction the sequencer provides a 12-bit next address from one of four sources : 1) the microprogram address counter (μPC), which usually contains an address one greater than the previous address, 2) an external (direct) input (D), 3) a register/counter (R) retaining data loaded during a previous microinstruction, or 4) a five deep last-in first-out stack (F). It has four instruction controls inputs (I). Under these instruction controls it can perform 16 different operations. $\overline{\text{CC}}$ and $\overline{\text{CCEN}}$ are two other inputs (conditional and conditional enable signals). These two signals can modify the instruction execution (conditional and unconditional branching). When the sequencer is used, it generates three output enable signals ($\overline{\text{PL}}$, $\overline{\text{MAP}}$, $\overline{\text{VECT}}$), which can be used to enable the outputs of pipeline register, mapping PROM and interrupt vector decoder output respectively.Only one of these outputs will be enabled at a time. (For details, refer to AMD Data Manual).

2.18 MICROPROGRAMME MEMORY (MM)

The MM has a capacity to store 4096 microinstructions. The size of each microinstruction (as developed in Chapter 3) is 64 bits. The MC sends a 12-bit address to the MM. It generates a 64-bit long microinstruction - which will be used to control all the control signals in this system in the next clock.

2.19 PIPELINE REGISTER (PR)

It is a 64-bit, positive edge triggered register. When its output is enabled with a control signal (PL), the 64-bit microinstruction control bits (PL) will be available as a current microinstruction and will be used to control all the control signals in this computing system.

2.20 CONDITION CODE MULTIPLEXER

It is used to increase the number of effective test conditions, to enrich the various test conditions for presentation to the Microprogramme Controller condition code input. The 16 inputs to the multiplexer are controlled by four signals. Depending upon the various status condition of the RALU and interrupt request signal, the MC can branch out to the corresponding microprogram for execution.

2.21 BUS CONTROL

Only two system buses are present in this system.They are Data and Address Buses (DB and AB). Data between various building blocks can be routed through this Data Bus. System Memory and DOB are connected to this DB. Only one control signal is therefore required to control the data flow to this DB.

PC, AR, and System Memory are connected with this AB. Since PC and AR can only address the System Memory, therefore, one control signal is required to enable one of these two logical building blocks onto the AB.

Each bus is 16-bit wide.

2.22 TIMING DIAGRAM

The system timing diagram is shown in Fig. 2.7.

Fig. 2.7.a Is the system clock (CP). The one cycle of the clock is called as one microcycle.

Fig, 2.7.b Is the Pipeline Register output, which contains 64-microinstruction controls bits. The timing of all the Logical building blocks will be refferred with respect to this output.

Fig. 2.7.c It shows the timing diagram of the address registers of the RALU, i.e., A,B address.

Fig. 2.7.d It indicates the local RAM output. When CP is high - RAM output data is stable (is RAM output A and RAM output B), when CP is low - RAM output data will be changing.

Fig. 2.7.e Indicates A,B latch output of RALU. When CP is high, the data in the latch is changing and when CP is low, the data in the latch will be stable.

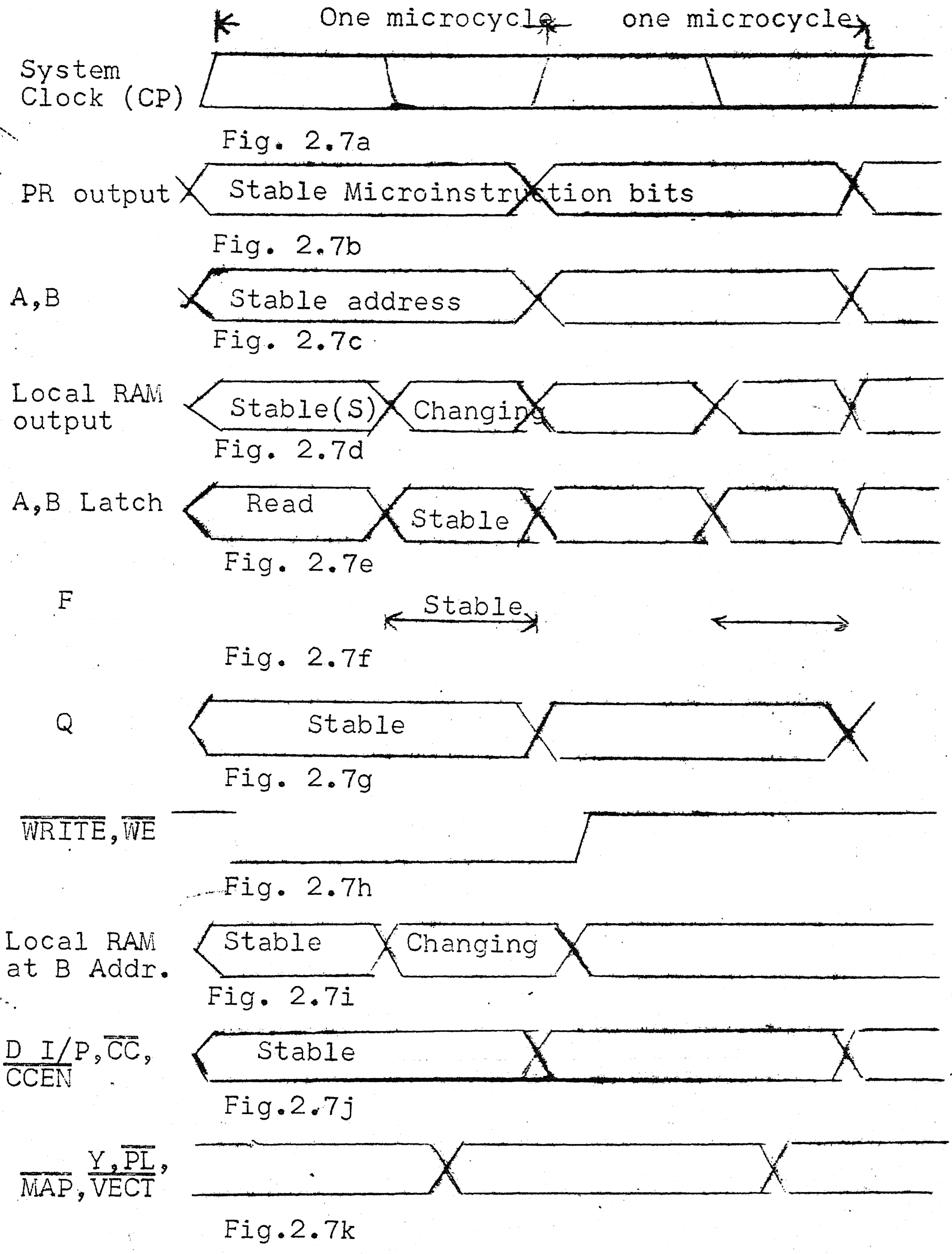


Fig. 2.7 System Timing Diagram

Fig. 2.7.f When CP is low, RALU performs its operation arithmetic, logic, special operation etc.) and result will be available after its operation is over.

Fig. 2.7.g Indicates the result of the RALU operation which can be latched onto the Q register when CP goes low to high.

Fig. 2.7.h Shows $\overline{\text{WRITE}}$ output and $\overline{\text{WE}}$ input signal. It generates when RALU performs Destination Operation.

Fig. 2.7.i Indicates RALU output (Y) data, can be written onto the Local RAM, addressed by B-address, when CP, $\overline{\text{WE}}$, $\overline{\text{WRITE}}$ signals are low. Data become stable when CP is high.

Fig. 2.7.j Indicate the D-input, Control signal to Am 2910(I) and condition code input signals.

Fig. 2.7.k Indicates the output of Am 2910 (Y). When $\overline{\text{OE}}$ is enabled. It also indicate $\overline{\text{PL}}$, $\overline{\text{MAP}}$ and $\overline{\text{VECT}}$ enable signals.

Pipeline Register, Macro Status Register and Address Register will be latched when CP goes low to high.

Till at this point, the various functional blocks of the system and their associated control signals have been discussed. The microinstruction which gives these control signals will be discussed in the next chapter.

CHAPTER 3

MICROPROGRAMME DEVELOPMENT

Bit slice devices are building blocks and need not be used with any particular type of control logic, but, they are normally discussed in the context of microprogrammed control logic. In fact, many of the available bit slice devices were designed to be used in microprogrammed control section. Because of this an understanding of microprogramming is needed to fully appreciate the nature of bit-slice logic.

A microprogramme is a technique for designing and implementing the control function of a data processing system as a sequence of control signals (microinstruction), to interpret fixed or dynamically alterable data processing functions. These control signals, organised on a word basis and stored in a fixed or dynamically alterable control memory, represent the states of the signals which control the flow of information between the executing functions and the orderly transition between the signal states.

A microinstruction specifies the steps comprising the machine sequence, directs the routing of data through the system and controls the parallel operation of the ALU.

In this chapter a discussion on the development of microprogramme, and followed by specifying and formating the

the microinstruction of the proposed system have been evolved. Microinstruction design entails the specification of the format of the microinstruction but also decisions concerning decoding logic, design of control section and system timing.

3.1 MICROORDER

A microinstruction word is divided or partioned into well defined sub-words called field or microorder. These micro-orders are choosen, such that, the interaction within any microorder is maximum and interaction among microorders are at a minimum. This is done by choosing functionally independent controls as a group and to allocate a microorder for each of them to specify a particular function.

3.2 MICROINSTRUCTION PIPELINING

Probably the most useful and powerful design idea in a microprogrammed system is the concept of microinstruction pipelining (sometimes referred as parallel implementation). It is a technique of allowing the control and processing sections of a processor to operate in parallel, such that the next microinstruction is being addressed and fetched in parallel with the control activities of the current microinstruction, thus shortening the machine cycle time.

Four commonly used pipelined structures are shown in Fig. 3.1.

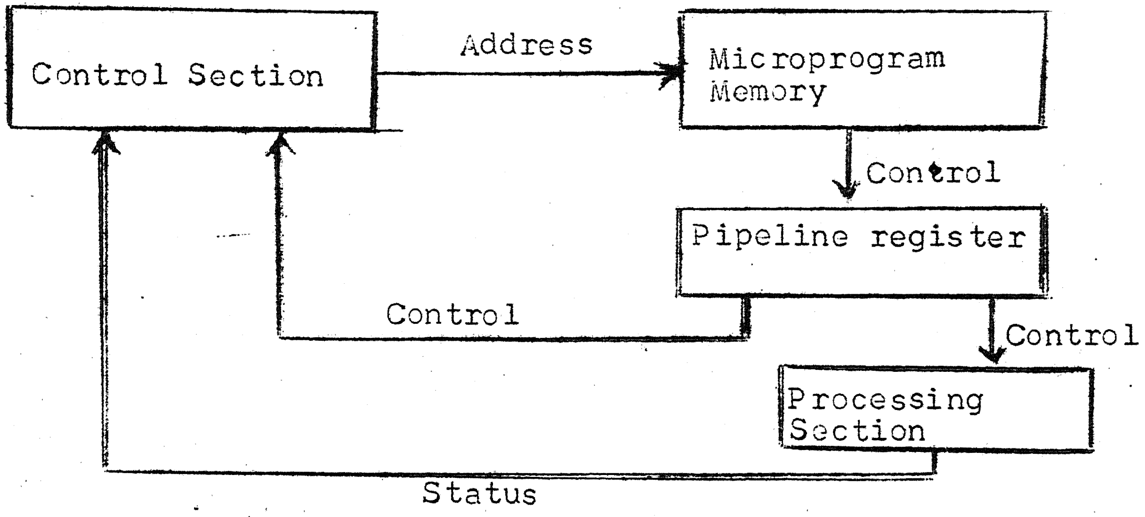


Fig. 3.1.a Instruction based

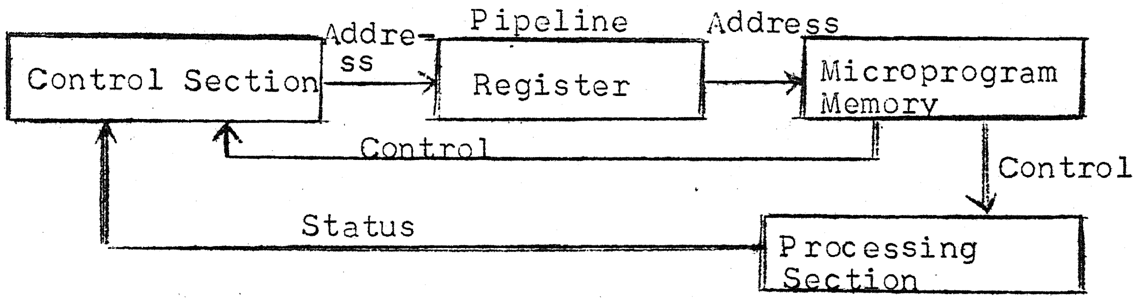


Fig. 3.1.b Addressed based

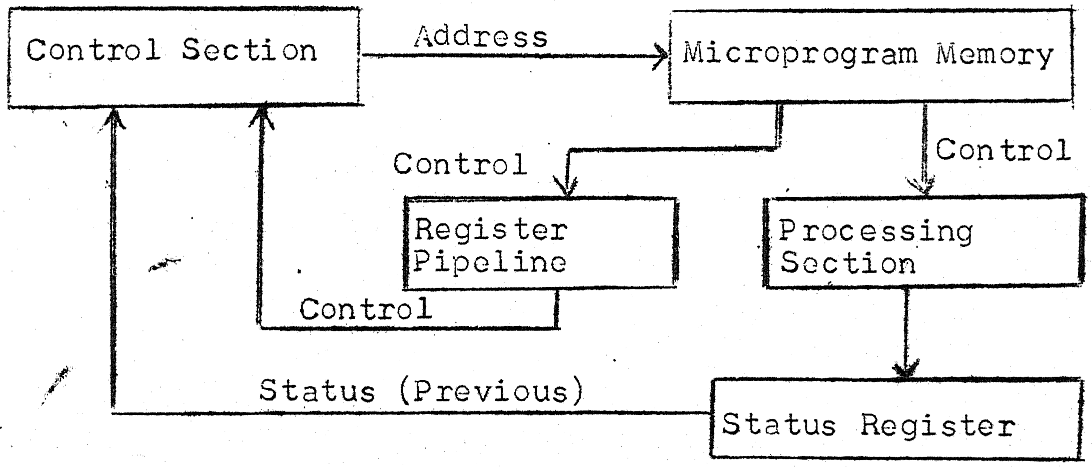


Fig. 3.1.c Data based

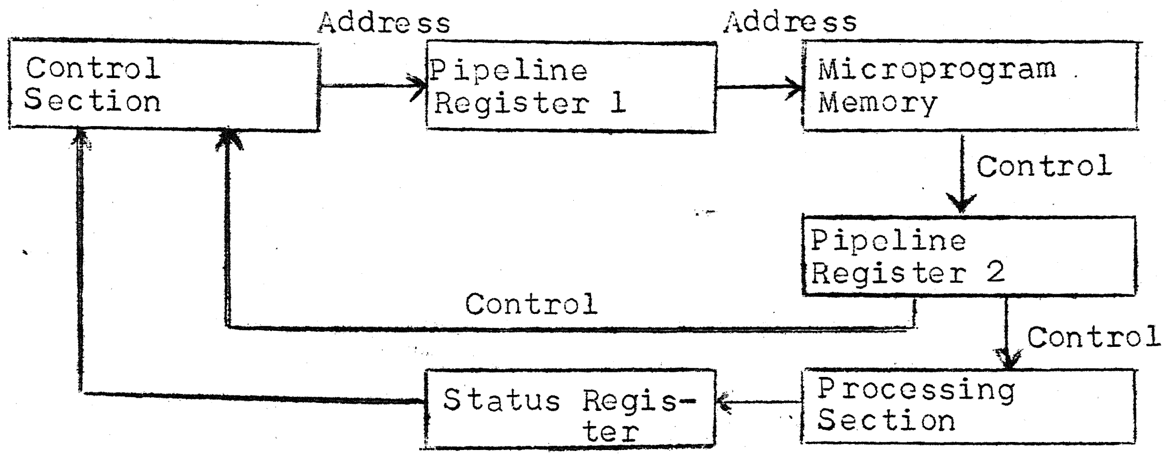
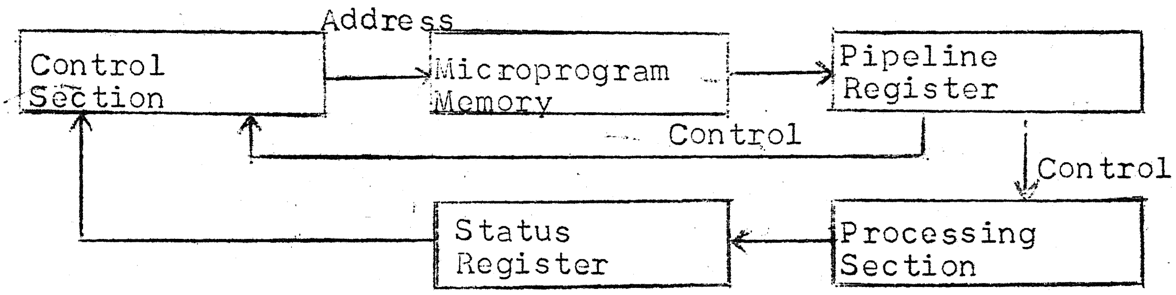
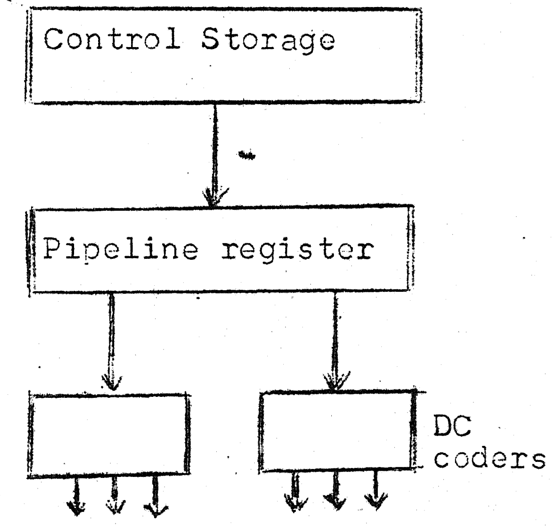
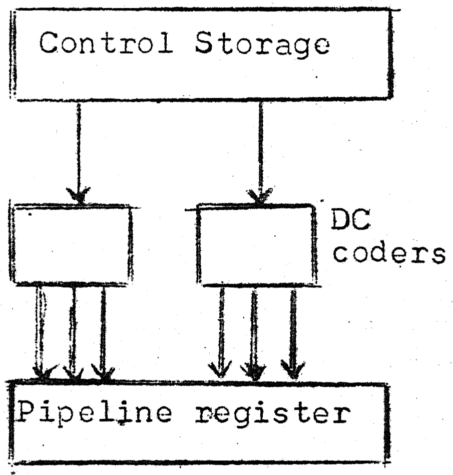


Fig. 3.1.d Two level Pipeline based





(a) Pre-Pipeline decoding



(b) Post-pipeine decoding

Fig. 3.3 Pipeline Decoding

In Instruction based -

The microprogramme memory and processing section's delay are in series. Conditional branches are executed on same cycle as the processing section generates the condition.

In Addressed based -

It provides about the same speed as in 1, but requires fewer register bits, since only the address is stored instead of microinstruction.

In Data based -

The status register provides conditional branch control based on results of previous processing cycle. The micro-program memory and the processing section are in series in the critical paths.

In Two level Pipeline based -

Two level pipeline provides highest possible speed. It is more difficult to program because the selection of a micro-instruction occurs two instructions ahead of its execution.

However, one level pipeline provides better speed than most other architecture. The microprogram memory and the processing section are in parallel speed paths instead in series. So the Fig. 3.2 architecture is the recommended approach for Am 2900 series design.

3.3 MICROORDER ENCODING

The encoding of control information in the microinstruction is usually motivated by two factors -

1) Reducing the width of the microinstruction. Hence the size of control storage.
2) Reducing the possibility of coding meaningless or erroneous microinstructions, i.e., specifying two functions that are truly mutually exclusive.

Since fast and large control-storage (memory) components are now available, concerning about control-storage space is now less critical than in they were in the past.

High degrees of encoding save control-storage space, but they increase the cost of decoding logic and invariably results in slower cycle times because of the added delays through the decoding logic. Therefore, limited amounts of encoding in extremely fast systems and in systems with a small number of control storage words is proposed. One technique is to use little or no encoding on microorders that appear in the critical timing paths, and employ higher degrees of encoding on microorders not on the critical timing path.

3.4 PRE AND POST PIPELINE DECODING

Pre and Post pipeline decoding helps keeping in a lower minimum machine cycle time. Appropriate placement of the decoding logic, one may be able to save 5 to 10% from the machine cycle time. Decoding can be done before or after the pipeline register as shown in Fig. 3.3.

If the processing section path is the longer path, and there is some decoding logic on this critical path, the machine cycle time can be reduced by placing the decoding logic between the control storage and the pipeline register. The trade off is that this requires a pipeline register of perhaps considerably more width. In some situations, it may be advantageous to mix these two approaches.

3.5 HORIZONTAL AND VERTICAL MICROINSTRUCTIONS

These are described in the context of the shape and size of control storage in a machine. In a machine with a horizontal type microinstruction, the control storage, relatively speaking, tends to be wide and shallow, whereas, vertical type microinstruction, control storage tends to be relatively narrow (short words) and deeper (more words).

A horizontal microinstruction contains a large number of independent microorders which exercise control over individual parts of the data flow. In other words it exhibits a high degree of parallelism because of its microorder. A highly vertical microinstruction contains relatively few fields and it is highly encoded,i.e.,itinvolves little or no parallesim within the machine cycle.

Advantages of using horizontal and vertical microinstruction in a machine cycle are -

- A branching operation can be specified by one or more microorders in a microinstruction and branching operation can be performed in one microcycle. In a vertical design, branching operations are usually not performed in parallel with control operations. Rather, each microinstruction simply sequences to the next microinstruction in control storage. Whenever, a branching operation is needed, it is performed in a separate cycle.

- In a higher-speed systems usually have horizontal designs, and slower-speed systems usually have vertical designs. This is due to (1) vertical microinstruction is more highly encoded, meaning that there is more overhead in the microinstruction decoding process, (2) large number of microorders in a typical horizontal microinstruction means that large number of operations can be performed in parallel in a single machine cycle, (3) in a branching microinstruction in most vertical design wastes a machine cycle whereas in the horizontal machine, branching operations are performed in parallel with control and processing operations.

Of course, in vertical machine would usually require less control storage space for a given microprogram than the horizontal machine.

3.6 MICROPROGRAMMING

The main advantage of microprogramming is that a microprogrammable machine can be used to emulate any of its subset machines. A typical microinstruction is a bit pattern of several parts. This can be usually classified into four broad fields -

(a) RALU field

(b) Next address control field or microprogram controller field

(c) Data routing and other control field

(d) Register address field.

With the appearance of microprogrammable bit-slice processors, the concept of user microprogrammability has gained popularity over the past few years. The ability of a user to write his own microprogramme for his specific application has many advantages over the conventional hardwired machine. He can produce a machine that is not only efficient but also conceptually simple. In the conventional hardwired or preprogrammed approach, the instruction set, once fixed, cannot be altered. Thus for a dedicated application, the user is unable to exploit the specific nature of his problem.

Microinstruction design is an optimization process involving goals concerning system cost, flexibility, and speed.

In performing this type of design, one balances such variables as the depth and width of control storage, clocking schemes and speeds, microprogramme branching flexibility, and the complexity and overhead of microinstruction decoding logic.

3.7 OP-CODE FORMAT FOR THE SYSTEM

Under the control of the microinstruction bit 40, the sixteen-bit OP-code is clocked onto the Instruction Register. The most significant 8-bit of the OP-code give the starting address of the microprogramme to be executed corresponding to the OP-code. The next 8-bits give the A and B addresses of the scratch pad register to be manipulated during the execution of a OP-code. The OP-code format is :

15 8	7 4	3 0	Bit No.
OP-CODE (Starting Address)	B3 B2 B1 BO Destination field	A3 A2 A1 A0 Source field	

As only B is the writing address of the scratch pad in the RALU, we will always indicate the destination field by the B-address, and the source field by the A-address.

The A and B addresses as part of the OP-code, it enables the user to access and manipulate any of the 16-scrach pad memory locations of the RALU.

3.8 MICROINSTRUCTION FORMAT

Perhaps the best way to review the design is to simply understand the function of each of the microinstruction control bits. It will help in understanding the design of the simple microcomputer CPU presented here.

The microinstruction for the proposed system is 64-bits wide. The functions of the microinstruction control bits are as follows :

RALU field

ALU requires 9 control signals to select the ALU source, function and destination. These nine control signals can be partitioned into two subwords.

Bits 1 to 4 - as control signal I_1-I_4 of the ALU is used for ALU functions.

Bits 5 to 8 - as control signal I_5-I_8 of the ALU, is used for ALU destination control.

Bit 9 - as control signal I_0 of the ALU.

Bits 10,11,12 - are devoted to be used as $\overline{\text{IEN}}$, $\overline{\text{EA}}$, $\overline{\text{OEB}}$ control signals.

These 3 bits (9,11,12) are grouped into one and used as ALU operand sources.

Bits 13,14,15,16 - This 4-bit wide field can be used either for the A-address, for the B-address or for both A and B address of the local RAM of the Am 2903.

Bits 17,18,19,20	-	4-bit wide field is used exclusively to address A of the local RAM of the Am 2903.
Bits 21,22	-	Select Am 2903 A-address source according to the table below.

Bits 22 21	A-Address Source
0 0	IR bits 0 through 3
0 1	microbits 17 through 20
1 0	IR bits 4 through 7
1 1	microbits 13 through 16

Bits 23	-	B-address field of the Am 2903 can select its address source from either IR or microbits, according to the table below.

Bits 23	B-Address Source
0	IR bits 4 through 7
1	microbits 13 through 16

Bits 24,25,26	-	These three bits and associated with (I_8) i.e. microinstruction bit 8 select the source for SIO_0, SIO_n, QIO_0 and QIO_n, for shift and rotate operation.

The following table summarizes the functions of these bits.

Microinstr. 26	25	Bits 24	SIO_n (Shift down)	SIO_0 (Shift-up)	QIO_n (Shift down)	QIO_0 (Shift-up)
0	0	0	0	0	0	0
0	0	1	SIO_0	SIO_n	QIO_0	QIO_n
0	1	0	QIO_0	QIO_n	SIO_0	SIO_n
0	1	1	Carry	QIO_n	SIO_0	Carry
1	0	0	Zero	Sign	SIO_0	SIO_n
1	0	1	Sign	Sign	Sign	Sign
1	1	0	N.A.	N.A.	N.A.	N.A.

*N.A. - not allotted.

The bit I_8 is used to decide up/down shift/rotate.

Bit 27 - It enables the clock to latch the status (macro or PSW) onto the macro status register.

Bit 28 - Is used with 60,61 and 62 bits to determine a particular status (either Micro or Macro) for the $\overline{CC}$ of Am 2910.

Bit 29 - MUX3, selects either present ALU status or macro status register output (i.e., delayed status) to CCM.

Bits 30,31 - Decoded to select C_n input of the least significant slice of Am 2903, according to the table below.

Microinst. 31	Bits 30	C_n input of LSS
0	0	0
0	1	1
1	0	From previous Carry
1	1	Zero

Bit 32 - When low, Am 2903 Y output is enabled and when it is high, Y-output is in tri-state.

<u>Data Routing and other Control</u> (PC, Interrupt) Field

Bits 33,34 - Selects the input to data out-buffer from the multiplexer 6 inputs according to the following table.

Microinst. 34	bits 33	Input to Data out -buffer
0	0	PC output
0	1	Status register output
1	0	Am 2903 (Y-bus) output
1	1	Not allotted

Bit 35	- It enables the clock to increment the program counter .
Bit 36	- Load control signal to program counter (PC). It enables the clock to load the Am 2903 output (Y-bus) to PC.
Bit 37	- Selects either memory read or write. This also controls output of the Data out buffer. When 0 - memory write and data out buffer output is enabled. When 1 - memory read and the output of the data out buffer is disabled. (Note - Data can be written onto the main memory when the chip-select of the memory and memory write control signal both are present).
Bit 38	- This is instruction enable input control signal to the interrupt controller. When it is low, the command on the instruction lines is executed and is ignored if it is high. When it is low the microprogram bits 12,24,25, 26 together control the four bit instruction field of the Am 2914 and ignored if it is high.

Microprogramme Controller field

Bit 40	- When low, enables the instruction register clock. The data present at bits 0 through 15 of the Data-Bus will be latched into the IR at the next low-to-high transition of the clock pulse.
Bits 41,42	- These are the $\overline{\text{RLD}}$ and $\overline{\text{CCEN}}$ control inputs of the Am 2910 sequencer respectively.
Bit 43	- This is the C1 input of the Am 2910 microprogram sequencer. When bit 43 is 1 - increments the sequencer output value. When bit 43 is 0 - loads non-incremented sequencer output to microprogram counter.
Bits 44,45,46,47	- Are the four I inputs of the Am 2910 sequencer.
Bits 48 through 59	- This is a 12-bit wide field and it serves, usually as the next microprogram address.
Bits 60,61,62 and 28	- These select the condition code input ($\overline{\text{CC}}$) according to the following table :

Microprogram Bits				Condition code selected
28	62	61	60	
0	0	0	0	Carry
0	0	0	1	Zero
0	0	1	0	Sign
0	0	1	1	Overflow
0	1	0	0	Interrupt request
0	1	0	1	SIO_n
0	1	1	0	SIO_0
0	1	1	1	QIO_0
1	0	0	0	Carry
1	0	0	1	Zero
1	0	1	1	Sign
1	0	1	1	Overflow
1	1	0	0	SIO_n
1	1	0	1	SIO_0
1	1	1	0	QIO_n
1	1	1	1	QIO_0

3.9 REGISTER ALLOCATION

In Am 2903 RALU is having sixteen general purpose registers/scratch pads. It is observed that usually six to eight resistors are quite adequate for scrach pad purposes. Therefore, the other resistors can be used for different purposes. Microprogramme designer can use these resistors with some preset values, which can be used in the microprogramme to manipulate data in such a way to minimise the number of microcycle. These resistors with preset values are transparent to the user. In this system the general purpose resistors are allocated in the following way and their functions are stated below :

Register No.	Preset data/Function
F	Preset to all '0's
E	Preset to all '1's
D	Reserved for future use
C	
B	Internal PC
A	Data Counter
9	Index Register
8	Stack Pointer
7	Used as a general
:	Purpose register
0	

3.10 IMPLEMENTATION OF INTEL 8085A INSTRUCTIONS

Popular Intel 8085A instruction sets are microprogrammed for this system. As example only five instructions are described here. Basically the data routing are stated here. The exact bit configurations for these examples can be found in Appendix II.

EXAMPLE 1 : AND data

MACRO instruction : ANI data ;

Addressing mode : IMMEDIATE ;

Operation : $(A) \leftarrow (A) \wedge (\text{word } 2)$.

Sequence of Operations :

a) OP Code fetch cycle (Fetch OP code from main memory and clock onto instruction register)./Continue/

b) PC $\rightarrow$ AB ; Select memory; AB $\rightarrow$ MEM ; Read memory (MEM) Content ; MEM $\rightarrow$ DB ; DB $\rightarrow$ DIR $\rightarrow$ DA $\rightarrow$ R ; Select RAM output B ; RAM output B $\rightarrow$ S ; Perform ALU operation (R AND S) $\rightarrow$ Y ; Y is written onto the RAM addressed by RB i.e. accumulator; PC will be incremented in the next positive clock.

c) OP code fetch cycle.

Note :

1. It is assumed R_0 of RALU is the Accumulator.

EXAMPLE 2 : JUMP address

MACRO instruction : JMP addr.

Addressing mode : IMMEDIATE ;

Operation : (PC) $\leftarrow$ (word 2) ;

Sequence of Operations :

a) OP-Code fetch cycle/Continue.
b) PC $\rightarrow$ AB; Select MEM ; AB $\rightarrow$ MEM ; MEM $\rightarrow$ DB
DB $\rightarrow$ DIR ; DIR $\rightarrow$ AB ; AB $\rightarrow$ R ; Perform ALU Operation
($F = R + C_n$; when $C_n = 0$) ; F $\rightarrow$ Y ; In the next positive
transition of CP, Y will be loaded to PC.
c) OP-Code fetch cycle.

EXAMPLE 3 : MOVE r1, r2

Operation : (r1) $\leftarrow$ (r2) .

Sequence of Operations :

a) OP-Code fetch cycle/Continue/
b) $IR_{0-3} \rightarrow$ RA $\rightarrow$ A ; $IR_{4-7} \rightarrow$ RB $\rightarrow$ B ; RAM output A $\rightarrow$ R ;
Perform ALU operation ($F = R+C_n$; when $C_n = 0$) ; Enable
$\overline{\text{WRITE}}$; Select $\overline{\text{IEN}} = \overline{\text{OEY}} = 0$; F $\rightarrow$ Y; Y will be written to
RAM, addressed by B. Written data will be stable when CP
goes low to high. In the next positive Clock transition
of the CP, PC will be incremented, (PC $\rightarrow$ PC+1).
c) OP-Code fetch cycle.

EXAMPLE 4 : ADD memory

MACRO instruction	:	ADD M ;
Addressing mode	:	REG. INDIRECT.
Operation	:	$(A) \leftarrow (A) + (H)\ (L))$;

Sequence of Operations

a) OP-Code fetch cycle/Continue/

b) PL(13-16) $\rightarrow$ RB ; RB $\rightarrow$ B; RAM output B $\rightarrow$ latch B ; $\overline{OEB} \rightarrow 0$; RAM output B $\rightarrow$ DB ; DB $\rightarrow$ AR ; Next Positive transition of the CP DB will be latched to AR/Continue/

c) AR $\rightarrow$ AB ; Select MEM ; AB $\rightarrow$ MEM ; MEM $\rightarrow$ DB; DB $\rightarrow$ DIR; DIR $\rightarrow$ DA ; DA $\rightarrow$ R ; PL (13-16) $\rightarrow$ RB ; RB $\rightarrow$ B ; RAM output B $\rightarrow$ S ; Perform ALU operation ($F = R + S + C_n$, where $C_n = 0$) ; $\overline{IEN} = \overline{OEY} = 0$; Y $\rightarrow$ RAM (write accomplished into RAM address by B). Next Positive transition of the CP, increments PC.

d) OP-Code fetch cycle.

Note :

1. It is assumed that HL register is R_1 in RALU, and Accumulator as R_o in the RALU.
2. These R_1 and R_o can be selected either from IR bits or PL outputs. Here I have selected both from PL, i.e., microinstruction bits.

<u>EXAMPLE 5 :</u> PUSH resistor pair

Addressing mode : REGISTER INDIRECT

Operation : $((SP) - 1) \leftarrow (rP)$.

<u>Sequence of Operations :</u>

a) OP-Code fetch cycle/Continue/

b) The contents of SP is decremented -

PL(13-16)$\rightarrow$ RB ; RB$\rightarrow$ B; RAM output B$\rightarrow$S ;

PL(17-20)$\rightarrow$ RA ; RA$\rightarrow$ A; RAM output A$\rightarrow$R ;

Perform ALU Operation ($F = S-R-1+C_n$, where $C_n = 0$ and $R = 0$);

$\overline{IEN} = \overline{OEY} = 0$; F$\rightarrow$ Y ; Y is written onto RAM - addressed by B/Continue/

c) MEM is addressed by the decremented SP value -

Keep the B same as in last cycle; RAM output B$\rightarrow$AR;

$\overline{OEB} = 0$; Any general purpose register can be selected by A;

RAM output A$\rightarrow$ R ; Perform ALU operation ($F = R + C_n$, when $C_n = 0$) ; $\overline{OEY} = 0$; F$\rightarrow$ Y ; Y$\rightarrow$ DOM$\rightarrow$ DOB $\rightarrow$ DB ; In the next positive transition of the CP AR content will be latched. /Continue/

d) AR$\rightarrow$ AB ; AB$\rightarrow$ MEM ; Select MEM ; WRITE MEM.

e) OP-Code fetch cycle.

<u>Note</u>

It is assumed SP is R_8 in RALU and R_F of RALU is having a pre-set value of zero.

CHAPTER 4

SIMULATION FOR THE BIT-SLICE PROCESSOR Am 2903

This chapter is devoted to the development of computer programme in PASCAL on an INTEL Series III Microprocessor Development System (MDS) for the simulation of Am 2903 bit-slice microprocessor chip.

In the process of the development of a microcomputing system, one of the most important logical building block one has to develope is the Central Processing Unit (CPU). In this CPU a logical sub-block is the ALU Operation Management Unit (AOMU). This AOMU mainly consists of Am 2903s and other associated chips (Registers and Multiplexers).

The Am 2903 is a 4-bit microprocessor slice. In the proposed system, four such slices are cascaded by using a fast Carry Look Ahead Generator (Am 2902) to obtain a 16-bit micro-processor module. In this simulator, instead of, simulating one Am 2903 slice and performing the execution of the same programme four times (in a FOR LOOP), this 16-bit module is simulated as one unit. This helps in minimizing the run-time for the execution of the simulator programme. Therefore, it is assumed that four such slices are cascaded by using one Am 2902 chip (as recommended by AMD Data manual) and all the interconne-ctions between the chips are established properly. It is also assumed that the $\overline{\text{WRITE}}$ output signal of the LSS is connected to all the inputs of $\overline{\text{WE}}$ signal of other slices, so as to enable writing onto the internal RAM of the 16-bit module.

4.1 THE OVERALL FLOW CHART

The Am 2903 is capable of performing certain operations simultaneously through parallel processing; but as the simulator programme can handle one logical step at a time, it will perform the operations of the Am 2903 sequentially. Based on this, a flow chart for a 16-bit Simulator has been shown in Fig. 4.1. This Simulator has a number of Procedures, each procedure being designed to perform a specific operation. Then calling a procedure within another procedure (nesting), the simulation of four main tables of Am 2903 chip (namely Operand sources, ALU arithmetic/logic functions, Destination/Shift and Rotate functions, as well as Special functions) have been made. The name of each simulating tables and the corresponding procedure names are as follows :

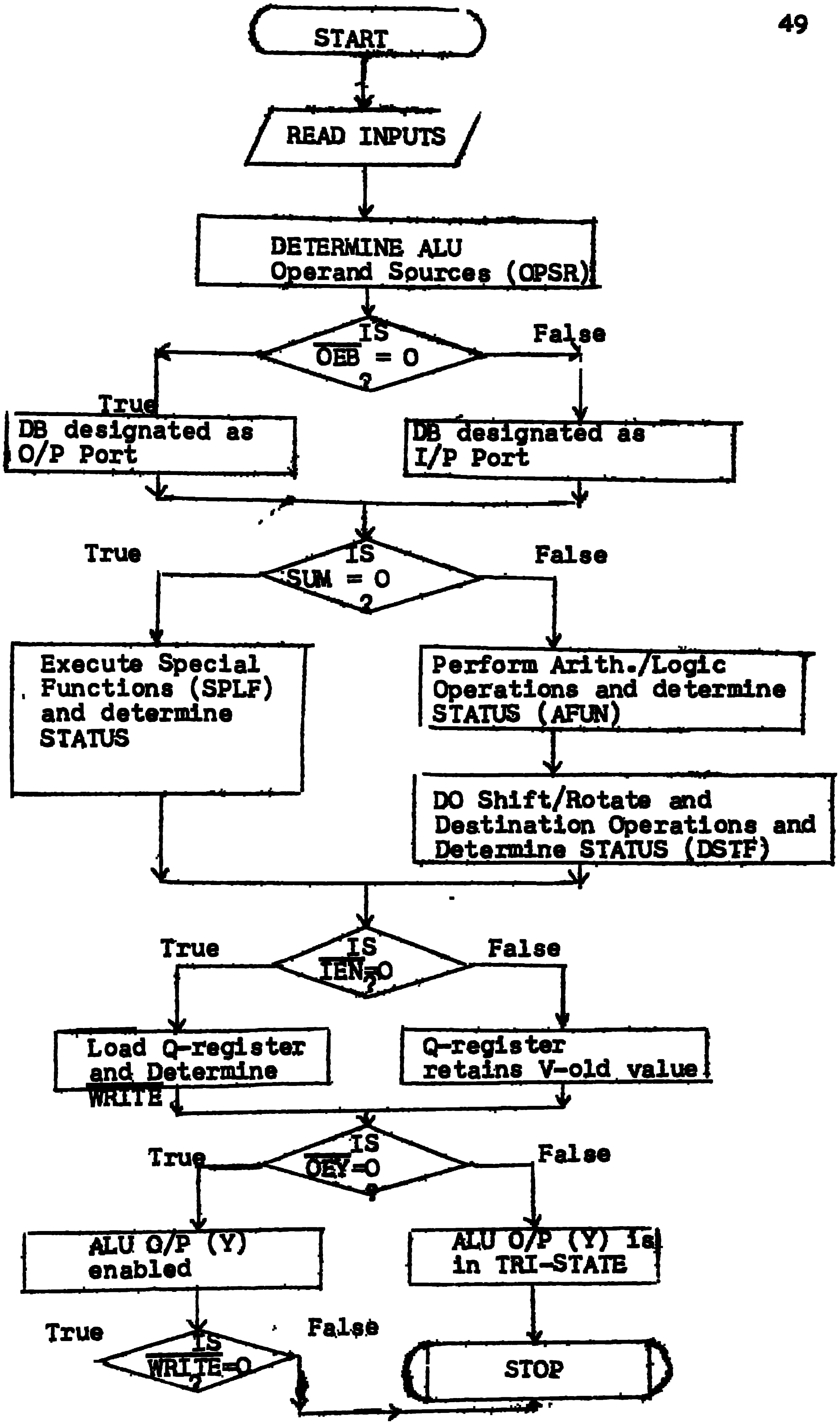
a) Operand Source Routine table (OPSR).

b) Arithmetic and Logical Function table (AFUN).

c) Destination, Shift/Rotate Function table (DSTF).

d) Special Function table (SPLF).

In the main programme the procedures are called sequentially in the same order as it would have executed the various functions within the chip. In this simulation, only chip functions are simulated; Pin-to-Pin simulation is not attempted.


START
READ INPUTS
DETERMINE ALU Operand Sources (OPSR)
IS $\overline{OEB}$ = 0 ?
True
False
DB designated as O/P Port
DB designated as I/P Port
IS SUM = 0 ?
True
False
Execute Special Functions (SPLF) and determine STATUS
Perform Arith./Logic Operations and determine STATUS (AFUN)
DO Shift/Rotate and Destination Operations and Determine STATUS (DSTF)
IS $\overline{IEN}$=0 ?
True
False
Load Q-register and Determine WRITE
Q-register retains V-old value
IS $\overline{OEY}$=0 ?
True
False
ALU O/P (Y) enabled
ALU O/P (Y) is in TRI-STATE
IS $\overline{WRITE}$=0 ?
True
False
STOP

Initially the programme has to read various input data; which are required to process its functions. After reading the inputs, it assigns the data to all the controls of the chip. It then fetches the operands required to perform the arithmetic and logical functions. (In the simulator, it is named as Procedure OPSR.) By checking the input datas, it will assign the DB, bidirectional I/O port, accordingly. If the SUM of the I_0 to I_4 variable input datas is equal to zero - it executes special function table (in the simulator, it is known as Procedure SPLF). If the sum is not equal to zero, then it will execute arithmetic and logical function table (known here as a AFUN). While executing these tables, it generates status outputs simultaneously. In special function, the destination and rotate/ shift functions are built in within the same procedure routine. Whereas in normal arithmetic, logical functions, the destination,shift and rotate operations are performed immediately after the AFUN procedure is over. If $\overline{\text{IEN}}$ (instruction enable I/p) is zero, it can then load the Q-register when clock goes low to high, otherwise it will retain its old values.

It checks the input data of $\overline{\text{OEY}}$ (output enable signal to Y) variable signal. If it is true, the result of the shifter output is enabled onto the Y (output of Am 2903), else the shifter output cannot be brought outside,i.e., on Y, and it will be tristated. At the time of performing the shift operation, it generates a $\overline{\text{WRITE}}$ output signal. If $\overline{\text{OEY}}$ is true and $\overline{\text{WRITE}}$ output is also true, then only internal RAM can be written with the data on Y, when clock remains low.

The behaviour of a 16-bit module is programmed for one clock cycle. For more than one clock cycle, this programme has to be executed repeatedly. Throughout the processing section the data are maintained in binary code. It helps inspecting and finding errors in every stage of operation and consequently it can be loaded onto the next block for further processing. The listing of the Am 2903 simulator programme is given in Appendix II.

4.2 INPUT REQUIREMENTS

This 16-bit module as a part of a microcomputing system, it expects, the principal inputs (control signals to the Am 2903s) to this Simulator are the microinstruction bits. Since only the Am 2903 is simulated here, therefore, to run this programme the user has to supply the data corresponding to the following control signals whenever the programme demands it. The control signals are :

I_{0-8}, A,B - addresses, DA, DB, $\overline{\text{EA}}$, $\overline{\text{OEB}}$, C_n,SIO_{15},SIO_0, QIO_{15}, QIO_0, $\overline{\text{IEN}}$ and $\overline{\text{OEY}}$.

The 16-general-purpose registers in these chips must be loaded with appropriate data before the OPSR procedure can be called in the main programme. For loading onto the RAM a procedure, RAMWRITE routine, is called in the main programme. For 16 RAM locations, the corresponding and appropriate data can be loaded onto the RAM by using this procedure routine.

4.3 STATUS

In this simulation the overflow and sign status are generated by using ripple carry algorithm, instead of calculating C_{n+4} and C_{n+3} for all individual bi-slices and then finally calculating overflow by using EX-OR operation between C_{n+4} and C_{n+3} for the MSS. This is because, it is assumed in the beginning that the simulation is for a 16-bit microprocessor module rather than 4x4-bit microprocessor slice.

The expressions used for calculating overflow and sign status from the two 16-bit operands (R and S) are as follows :

$$\mathrm{OVR} = \overline{S}_R . \overline{S}_S . S_F + S_R . S_S . \overline{S}_F$$

and

$$\mathrm{SIGN} = S_R . S_F + S_S . S_F + S_R . S_S$$

where $\overline{S}_R$ is the complement of S_R and S_R denotes the sign bit (MSB) of the R operand. Similarly, $\overline{S}_S$ is the complement of S_S and S_S denotes the sign bit (MSB) of the S operand. $\overline{S}_F$ is the complement of S_F and S_F itself is denoted as sign bit (MSB) of the ALU output (F).

Table 4.1

ALU Operand Sources

$\overline{E}_A$	I_0	$\overline{OE}_B$	ALU Operand R	ALU Operand S
L	L	L	RAM Output A	RAM Output B
L	L	H	RAM Output A	DB_{0-3}
L	H	X	RAM Output A	Q Register
H	L	L	DA_{0-3}	RAM Output B
H	L	H	DA_{0-3}	DB_{0-3}
H	H	X	DA_{0-3}	Q Register

L = Low H = High X = Don't Care

Table 4.2

Am 2903 ALU Functions

I_4	I_3	I_2	I_1	Hex Code	ALU Functions	
L	L	L	L	0	I_0 = L I_0 = H	Special Functions F_1 = High
L	L	L	H	1	F = S Munus R Minus 1 Plus C_n	
L	L	H	L	2	F = R Minus S Minus 1 Plus C_n	
L	L	H	H	3	F = R Plus S Plus C_n	
L	H	L	L	4	F = S Plus C_n	
L	H	L	H	5	F = $\bar{S}$ Plus C_n	
L	H	H	L	6	F = R Plus C_n	
L	H	H	H	7	F = $\bar{R}$ Plus C_n	
H	L	L	L	8	F_1 = LOW	
H	L	L	H	9	$F_i = \bar{R}_1$ and S_i	
H	L	H	L	A	$F_i = R_i$ EXCLUSIVE NOR S_i	
H	L	H	H	B	$F_i = R_i$ ESCLUSIVE OR S_i	
H	H	L	L	C	$F_i = R_i$ and S_i	
H	H	L	H	D	$F_i = R_i$ NOR S_i	
H	H	H	L	E	$F_i = R_i$ NAND S_i	
H	H	H	H	F	$F_i = R_i$ OR S_i	

L = LOW H = HIGH $i = 0$ to 3

I_8	I_7	I_6	I_5	Hex Code	ALU Shifter Function	SIO_3		Y_3		Y_2		Y_1	Y_0	SIO_0	$\overline{Write}$	Q Reg & Shifter Function	QIO_3	QIO_0
						Most Sig. Slice	Other Slices	Most Sig. Slice	Other Slices	Most Sig. Slice	Other Slices							
L	L	L	L	0	Arith $F/2 \rightarrow Y$	Input	Input	F_3	SIO_3	SIO_3	F_3	F_2	F_1	F_0	L	Hold	Hi-Z	Hi-Z
L	L	L	H	1	Log $F/2 \rightarrow Y$	Input	Input	SIO_3	SIO_3	F_3	F_3	F_2	F_1	F_0	L	Hold	Hi-Z	Hi-Z
L	L	H	L	2	Arith $F/2 \rightarrow Y$	Input	Input	F_3	SIO_3	SIO_3	F_3	F_2	F_1	F_0	L	Log $Q/2 \rightarrow Q$	Input	Q_0
L	L	H	H	3	Log $F/2 \rightarrow Y$	Input	Input	SIO_3	SIO_3	F_3	F_3	F_2	F_1	F_0	L	Log $Q/2 \rightarrow Q$	Input	Q_0
L	H	L	L	4	$F \rightarrow Y$	Input	Input	F_3	F_3	F_2	F_2	F_1	F_0	Parity	L	Hold	Hi-Z	Hi-Z
L	H	L	H	5	$F \rightarrow Y$	Input	Input	F_3	F_3	F_2	F_2	F_1	F_0	Parity	H	Log $Q/2 \rightarrow Q$	Input	Q_0
L	H	H	L	6	$F \rightarrow Y$	Input	Input	F_3	F_3	F_2	F_2	F_1	F_0	Parity	H	$F \rightarrow Q$	Hi-Z	Hi-Z
L	H	H	H	7	$F \rightarrow Y$	Input	Input	F_3	F_3	F_2	F_2	F_1	F_0	Parity	L	$F \rightarrow Q$	Hi-Z	Hi-Z
H	L	L	L	8	Arith $2F \rightarrow Y$	F_2	F_3	F_3	F_2	F_1	F_1	F_0	SIO_0	Input	L	Hold	Hi-Z	Hi-Z
H	L	L	H	9	Log $2F \rightarrow Y$	F_3	F_3	F_2	F_2	F_1	F_1	F_0	SIO_0	Input	L	Hold	Hi-Z	Hi-Z
H	L	H	L	A	Arith $2F \rightarrow Y$	F_2	F_3	F_3	F_2	F_1	F_1	F_0	SIO_0	Input	L	Log $2Q \rightarrow Q$	Q_3	Input
H	L	H	H	B	Log $2F \rightarrow Y$	F_3	F_3	F_2	F_2	F_1	F_1	F_0	SIO_0	Input	L	Log $2Q \rightarrow Q$	Q_3	Input
H	H	L	L	C	$F \rightarrow Y$	F_2	F_3	F_3	F_3	F_2	F_2	F_1	F_0	Hi-Z	H	Hold	Hi-Z	Hi-Z
H	H	L	H	D	$F \rightarrow Y$	F_3	F_3	F_3	F_3	F_2	F_2	F_1	F_0	Hi-Z	H	Log $2Q \rightarrow Q$	Q_3	Input
H	H	H	L	E	$SIO_0 \rightarrow Y_0, Y_1, Y_2, Y_3$	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	Input	L	Hold	Hi-Z	Hi-Z
H	H	H	H	F	$F \rightarrow Y$	F_3	F_3	F_3	F_3	F_2	F_2	F_1	F_0	Hi-Z	L	Hold	Hi-Z	Hi-Z

Parity = $F_3 \forall F_2 \forall F_1 \forall F_0 \forall SIO_3$ L = LOW Hi-Z = High Impedance

$\forall$ = Exclusive OR H = HIGH

Figure 20a. ALU Destination Control for I_0 or I_1 or I_2 or I_3 or I_4 = HIGH, $\overline{IEN}$ = LOW.

I_8	I_7	I_6	I_5	Hex Code	Special Function	ALU Function	ALU Shifter Function	SIO_3		SIO_0	Q Reg & Shifter Function	QIO_3	QIO_0	$\overline{WRITE}$
								Most Sig. Slice	Other Slices					
L	L	L	L	0	Unsigned Multiply	$F = S + C_n$ if $Z = L$ $F = R + S + C_n$ if $Z = H$	Log. $F/2 \rightarrow Y$ (Note 1)	Hi-Z	Input	F_0	Log. $Q/2 \rightarrow Q$	Input	Q_0	L
L	L	H	L	2	Two's Complement Multiply	$F = S + C_n$ if $Z = L$ $F = R + S + C_n$ if $Z = H$	Log. $F/2 \rightarrow Y$ (Note 2)	Hi-Z	Input	F_0	Log. $Q/2 \rightarrow Q$	Input	Q_0	L
L	H	L	L	4	Increment by One or Two	$F = S + 1 + C_n$	$F \rightarrow Y$	Input	Input	Parity	Hold	Hi-Z	Hi-Z	L
L	H	L	H	5	Sign/Magnitude-Two's Complement	$F = S + C_n$ if $Z = L$ $F = \bar{S} + C_n$ if $Z = H$	$F \rightarrow Y$ (Note 3)	Input	Input	Parity	Hold	Hi-Z	Hi-Z	L
L	H	H	L	6	Two's Complement Multiply, Last Cycle	$F = S + C_n$ if $Z = L$ $F = S - R - 1 + C_n$ if $Z = H$	Log. $F/2 \rightarrow Y$ (Note 2)	Hi-Z	Input	F_0	Log. $Q/2 \rightarrow Q$	Input	Q_0	L
H	L	L	L	8	Single Length Normalize	$F = S + C_n$	$F \rightarrow Y$	F_3	F_3	Hi-Z	Log. $2Q \rightarrow Q$	Q_3	Input	L
H	L	H	L	A	Double Length Normalize and First Divide Op.	$F = S + C_n$	Log $2F \rightarrow Y$	$R_3 \forall F_3$	F_3	Input	Log. $2Q \rightarrow Q$	Q_3	Input	L
H	H	L	L	C	Two's Complement Divide	$F = S + R + C_n$ if $Z = L$ $F = S - R - 1 + C_n$ if $Z = H$	Log. $2F \rightarrow Y$	$\overline{R_3 \forall F_3}$	F_3	Input	Log. $2Q \rightarrow Q$	Q_3	Input	L
H	H	H	L	E	Two's Complement Divide, Correction and Remainder	$F = S + R + C_n$ if $Z = L$ $F = S - R - 1 + C_n$ if $Z = H$	$F \rightarrow Y$	F_3	F_3	Hi-Z	Log. $2Q \rightarrow Q$	Q_3	Input	L

NOTES: 1. At the most significant slice only, the C_{n+4} signal is internally gated to the Y_3 output.
2. At the most significant slice only, $F_3 \forall OVR$ is internally gated to the Y_3 output.
3. At the most significant slice only, $S_3 \forall F_3$ is generated at the Y_3 output.
4. Op codes 1, 3, 7, 9, B, D, and F are reserved for future use.

L = LOW	Hi-Z = High Impedance
H = HIGH	$\forall$ = Exclusive OR
X = Don't Care	Parity = $SIO_3 \forall F_3 \forall F_2 \forall F_1 \forall F_0$

Figure 17. Special Functions: $I_0 = I_1 = I_2 = I_3 = I_4 = LOW$, $\overline{IEN} = LOW$.

CHAPTER 5

SIMULATOR OF A MICROPROGRAMME CONTROLLER Am 2910

Microprogramme Controller is another important logical building block, which will be used to design in the Micro-programme Management Unit (MMU). The other associated logical functional blocks used in the MMU are Registers and Memory. Since these associated chips are functionally very simple, therefore, the attention has been drawn to simulate a more complecated chip like Am 2910 in begin with.

The Am 2910 is a 12-bit wide Microprogramme Controller, it cannot be cascaded for a wider microprogrammable controller application. It is capable of performing certain operations i parallel in one cycle time. But due to the sequential nature the simulator programme, it is not possible to execute paralle operation simultaneously. Based on the sequential nature of t programme a flow chart for this simulator has been shown in Fig. 5.1.

5.1 DESCRIPTION OF THE SIMULATOR

This simulator consists of a number of procedures. Each procedures will perform a specific set of operations (like loading the register, decrement the register content, push o the stack,pop out from the stack etc.). The main program

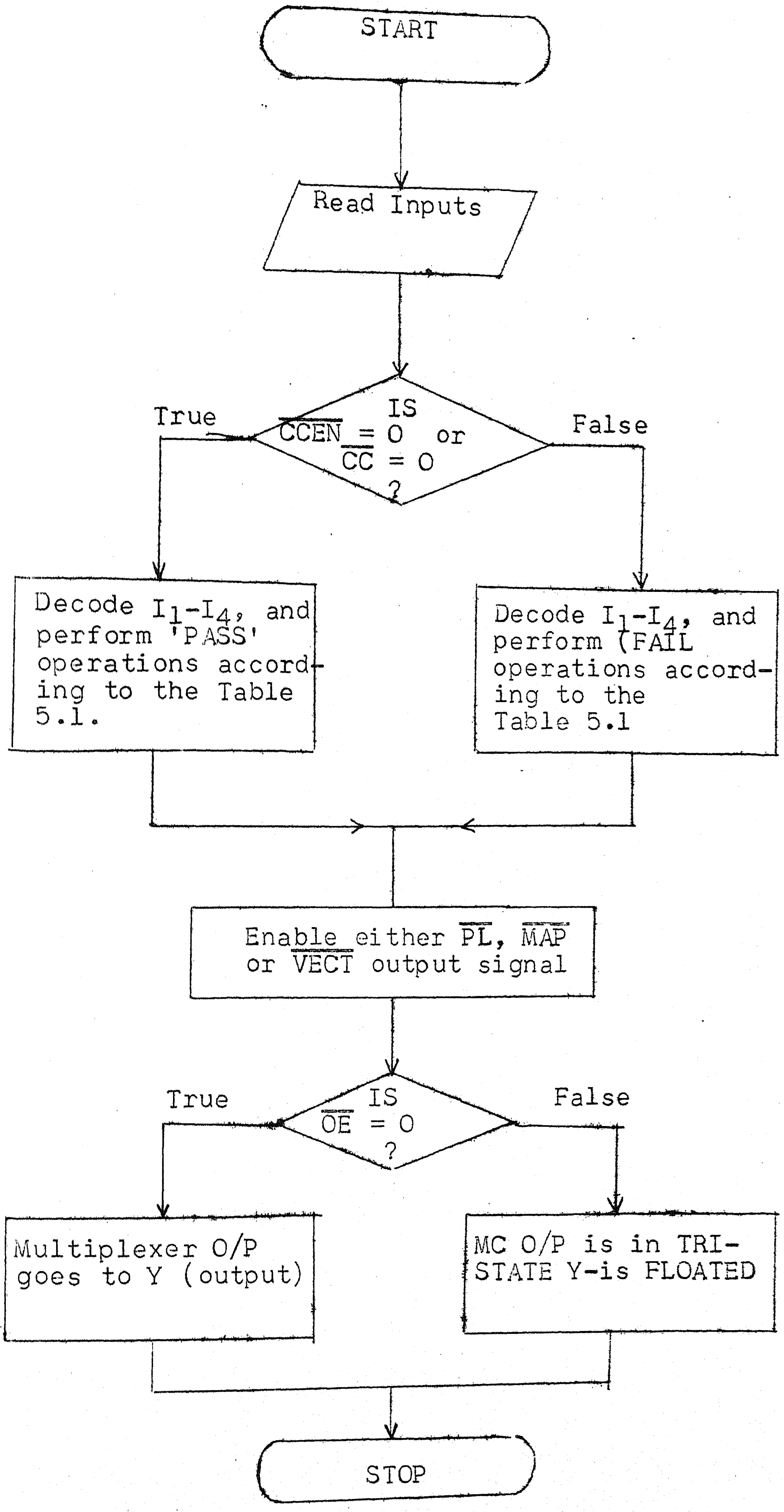

START
Read Inputs
IS $\overline{CCEN} = 0$ or $\overline{CC} = 0$?
True
False
Decode I_1-I_4, and perform 'PASS' operations according to the Table 5.1.
Decode I_1-I_4, and perform (FAIL operations according to the Table 5.1
Enable either $\overline{PL}$, $\overline{MAP}$ or $\overline{VECT}$ output signal
IS $\overline{OE} = 0$?
True
False
Multiplexer O/P goes to Y (output)
MC O/P is in TRI-STATE Y-is FLOATED
STOP

Fig. 5.1 Am 2910 Simulator Flow Chart

TABLE 4. Am2910 MICROINSTRUCTION SET.

HEX I_3-I_0	MNEMONIC	NAME	REG/CNTR CON-TENTS	FAIL $\overline{CCEN}$ = LOW and $\overline{CC}$ = HIGH		PASS $\overline{CCEN}$ = HIGH or $\overline{CC}$ = LOW		REG/CNTR	ENABLE
				Y	STACK	Y	STACK		
0	JZ	JUMP ZERO	X	0	CLEAR	0	CLEAR	HOLD	PL
1	CJS	COND JSB PL	X	PC	HOLD	D	PUSH	HOLD	PL
2	JMAP	JUMP MAP	X	D	HOLD	D	HOLD	HOLD	MAP
3	CJP	COND JUMP PL	X	PC	HOLD	D	HOLD	HOLD	PL
4	PUSH	PUSH/COND LD CNTR	X	PC	PUSH	PC	PUSH	Note 1	PL
5	JSRP	COND JSB R/PL	X	R	PUSH	D	PUSH	HOLD	PL
6	CJV	COND JUMP VECTOR	X	PC	HOLD	D	HOLD	HOLD	VECT
7	JRP	COND JUMP R/PL	X	R	HOLD	D	HOLD	HOLD	PL
8	RFCT	REPEAT LOOP, CNTR ≠ 0	≠ 0	F	HOLD	F	HOLD	DEC	PL
			= 0	PC	POP	PC	POP	HOLD	PL
9	RPCT	REPEAT PL, CNTR ≠ 0	≠ 0	D	HOLD	D	HOLD	DEC	PL
			= 0	PC	HOLD	PC	HOLD	HOLD	PL
A	CRTN	COND RTN	X	PC	HOLD	F	POP	HOLD	PL
B	CJPP	COND JUMP PL & POP	X	PC	HOLD	D	POP	HOLD	PL
C	LDCT	LD CNTR & CONTINUE	X	PC	HOLD	PC	HOLD	LOAD	PL
D	LOOP	TEST END LOOP	X	F	HOLD	PC	POP	HOLD	PL
E	CONT	CONTINUE	X	PC	HOLD	PC	HOLD	HOLD	PL
F	TWB	THREE-WAY BRANCH	≠ 0	F	HOLD	PC	POP	DEC	PL
			= 0	D	POP	PC	POP	HOLD	PL

Note: If $\overline{CCEN}$ = LOW and $\overline{CC}$ = HIGH, hold; else load. X = Don't Care.

was developed on the basis of Table 1, and they will be required to execute this simulation programme. The name of the procedures and their functions are stated below :

	Name of the Procedure	Function
1.	CLRSTACK	Content of the stack is made to zero.
2.	READSTACK	To read the content of the stack top only.
3.	POP	To Pop out the content of the stack top and decrement the stack pointer
4.	PUSH	The content of the microprogramme counter is pushed to the stack top.
5.	LOADREG	Register/Counter is loaded with its input data.
6.	CMPC	To increment the microprogramme counter content.
7.	DECREMENT	To decrement the register/counter output.

Throughout this simulatinn the data are maintained in decima number. It helps inspecting and finding errors in every stage of operation. The listing of Am 2910 Simulator Programme is gi in Appendix III.

5.2 DESCRIPTION OF THE FLOW CHART

This program has to read various input data, which are required to execute this programme. It then test the condition code input signal ($\overline{CC}$) if it is proved that the result of the test is positive, it decodes the I_1-I_4 input control signals and executes the corresponding 'PASS' condition programme. But if the result of the test is negative, it executes a 'FAIL' condition programme, corresponding to the decoded value of I_1-I_4 input control signals.

Maihly the programme will determine the output of the multiplexer from its four possible inputs. Once the output of the multiplexer is obtained it will update the microprogramm counter content either by incrementing or retaining the same value as the multiplexer output. The Stack, microprogramme counter, and register/counter operation are done with respect to the positive edge of the clock. The register/counter will either decrement or will retain its value. After these operations it will enable one of the following output $\overline{PL}$, $\overline{MAP}$, $\overline{VECT}$ It checks $\overline{OE}$ (output enable) input signal. If it is true, the multiplexer output will be enabled to the Y (output), otherwis the Y (output) will be floated.

5.3 INPUT REQUIREMENTS

This microprogramme controller, expects, the principal inputs to this simulator are the microinstruction bits. Sinc here, only the Am 2910 is simulated, to run this programme, a user has to supply the data corresponding to the following c trol signals, whenever, the programme demands for it.

The control signals are :

I_{1-4}, $\overline{CC}$, $\overline{CCEN}$, Cl, $\overline{RLD}$, $\overline{OE}$ and D_{1-12} .

The D input signals again can be either from MAPO (Mapping PROM output), or from VECA (Vector address from Vector Decoder) or from BA (microinstruction bits).

5.4 DESCRIPTION OF PROCEDURES

CLRSTACK

The stack in Am 2910 is a five-word last-in, first-out 12 bit memory, has a pointer which addresses the value presently on the top of the stack. Before performing the stack operation in the system, it requires initialization, to clear the content of all the locations in the stack, and make the stack pointer = 0, so that it can be ready for future use.

READSTACK

Sometimes in stack operation, it may require to read the stack top only, but not to decrement the stack-pointer. It helps in looping operation. This procedure will be active only, when the stack-pointer value is greater than or equal to one. Obviously when the stack pointer crosses the stacksize, the attempt should not be used to read the stack.

POP

In stack operation, the stack top can be popped out, provided the stack pointer value is greater than zero. After the pop operation, the stack pointer value will be decremented. If stack pointer points to zero, then further pop operation will not be possible and if it is attempted to pop out the content of the stack, it will warn the user by indicating that stack is empty and so it cannot pop.

PUSH

PUSH operation will be possible, if stack-pointer is less than the stack size (i.e., 5). It will, therefore, first check the pointer value. If it is within the stack size limit, it increments the stack-pointer value from its present value and pushes the content of the microcounter output to the stack, when clock goes from low to high. But if pointer value = stack size, then it gives warning to the user - stating that the stack is full and further push operation will destroy the previous stack top value. Therefore, user should avoid this condition.

LOADREG

In Am 2910 Register/Counter can be loaded with the D-input value by two ways. If the $\overline{\text{RLD}}$ control signal is zero, then irrespective of the instructions (I), it will load the register/

counter with the value of D-input. In this procedure, it will check $\overline{RLD}$ control signal, if it is zero, then only it will load onto the register/counter, otherwise it will ignore it.

CMPC

In this procedure,the incrementer and the microprogramme counter functions both are included. It checks the content of CI variable. If it is one it increments the multiplexer output value by one, if it is zero it maintains the same value of the multiplexer output. This CMPC updated value can be loaded onto the stack as well as it can be brought to one of the input to the multiplexer.

DECREMENT

Since in Am 2910, $\overline{RLD}$ (load to register/counter) function overrides any other register/counter operations, therefore,this procedures checks the $\overline{RLD}$ input variable first. If it is zero, it will perform only LOADREG function, otherwise it will decrement the content of the register/counter. If it is found that the register/counter content is already zero before decrementing, it gives a warning to the user by stating further decrement is not possible, because it has already reached a zero value.

The behaviour of a Am 2910 is programmed for one cycle. For more than one clock cycle, this programme has to be executed repeatedly.

CONCLUSION

Two programmes have been written to simulate in complete details the functions of the 16-bit processor module and an Microprogramme Controller. Complete Simulation needs inter-connecting programmes (small procedures for the different functional blocks and inter-connecting them in the main programme). All other hardware need to be simulated in functional level and as such it will be relatively simple. The very strength of the Bit-Slice Processor lies on the custom oriented configuration and hence the architecture suggested in this thesis can only give a guideline. It is expected the user should make its own architecture and using these Simulators will verify the micro-programmes.

As the interconnection between the modules (logical building blocks) have not been simulated, the programmes have been verified, by modifying the actual Simulator programmes into an interactive form with the following features.

These programmes need some input data information to execute the programmes (like, instructions I_0-I_8; Address A; Address B ; DA, DB, etc.). These data have to be supplied by the user in a particular format, whenever these programmes demand it. (Format will be defined in the programme). In case the user fails to respond properly, the programme will not advance further. It will ask for the input again - until the programmes satisfies with the inputs. Similarly after executing, it will display the result of an operation and wait for further action. In the end when the simulation will be over, it will ask, if the user wants to continue the simulation again for an another set of inputs or to quit from further processing.

Even these programmes have been tested with all possible inputs condition in a Loop. The execution results of such programmes are also listed.

REFERENCES

1. Katzan H.: 'Computer Systems Organization and Programming, Science Research Associates, Inc., 1976.
2. Advanced Micro Devices, 'Bipolar Microprocessor Logic and Interface Data Book.
3. Alexandridis N.A., 'Bit-sliced Microprocessor Architecture, COMPUTER, June 1978.
4. Mick and Brick, 'Bit-Slice Microprocessor Design.
5. Glenford J. Myers, 'Digital System Design with LSI Bit-Slice Logic.
6. Adam Osborne and Jerry Kane, 'An Introduction to Micro-computers', Vol.2, Part B.
7. Guy G. Boulaye, 'Microprogramming'.
8. A. Singh and A. Saronwala, 'Designing with Bit-Slice Microprocessors - Hardware Feasibility and Simulation Studies, B.Tech. Thesis, April, 1980, I.I.T. Kanpur.
9. Jensen K., Wirth N., 'PASCAL User Manual and Report'.
10. Rajaraman, V., 'Computer Programming in PASCAL'.

		2903								2903							PC			STATUS		2910					MEM			INTR
		I_{1-4}	I_{5-8}	I_0	$\overline{IEN}$	$\overline{EA}$	$\overline{OEB}$	$SIO_0/\overline{QIO_0}$	$SIO_n/\overline{QIO_n}$	Rdst	Rsou	A-Addr	B-Addr	C_n	$\overline{OEY}$	I/P-DOB	$\overline{INC}$	PC+AB	LOAD	MACRO	PSW/PREST	$\overline{RLD}$	$\overline{CCEN}$	CI	I	CC	$\frac{MR}{MW}$	$\overline{IR}$/DIR	PL	$\overline{IE}$
No. of Micro	Inst. BIT →	4	4	1	1	1	1	3	4	4	2	1	2	1	2	1	1	1	1	1	1	1	1	4	4	1	1	12	1	
Bit No →	Micro-Program Addr.	1-4	5-8	9	10	11	12	24-26	13-16	17-20	21-22	23	30-31	32	33-34	35	39	36	27	29	41	42	43	44-47	28, 8 60-62	37	40	48-59	38	
ANI data	01A	C	F	0	0	1	0	x	0	x	x	1	0	0	x	0	0	1	0	0	1	1	0	7	x	1	1	001	1	
JMP addr	02C	6	C	x	0	1	x	x	x	x	x	x	0	0	x	0	0	0	0	0	1	1	0	7	x	1	1	001	1	
MOV r1,r2	040	6	F	0	0	0	x	x	x	x	0	0	0	0	x	0	x	1	0	0	1	1	0	7	x	1	1	001	1	
ADD M	006	x	C	1	1	x	0	x	1	x	x	1	0	1	x	1	1	1	1	1	1	1	1	E	x	1	1	xxx	1	
	007	3	F	0	0	1	0	x	0	x	x	1	0	0	x	0	1	1	0	0	1	1	0	7	x	1	1	001	1	
PUSH rp	027	1	F	0	0	0	0	x	8	F	1	1	0	0	x	1	0	1	1	0	1	1	1	E	x	1	1	xxx	1	
	028	6	C	x	0	0	0	x	8	x	0	1	0	0	2	1	0	1	1	0	1	1	1	E	x	1	1	xxx	1	
	029	6	C	x	0	0	0	x	8	x	0	1	0	0	2	0	1	1	0	0	1	1	0	7	x	0	1	001	1	

N.B. • All fields in HEX
• X = Don't Care

Implementation of Intel 8085A Instructions

```
ES-III Pascal-86, V2.0


ce File:  :F3:ALUSIM
ct File:  :F3:ALUSIM.OBJ
rols Specified: <none>.

 NESTING          SOURCE TEXT:  :F3:ALUSIM
 0  0             PROGRAM  AM2903(INPUT,OUTPUT);
 0  0             TYPE
                       BARRAY=ARRAY[1..16] OF INTEGER;
 0  0                  MAT=ARRAY[1..16,1..16] OF INTEGER;
 0  0                  BOOL=ARRAY[1..16] OF BOOLEAN;
 0  0             VAR
                       SADDR,Q,DB,DA,DIN,ADDRA,ADDRB,F,TEMP:BARRAY;
 0  0                  I,INT,Y,QRIN,ASO,OD,R,S:BARRAY;
 0  0                  RAMDATA:MAT;
 0  0                  INSD,INP,J,CIN,N,SIO3,SIO0,QIO3,QIO0,OEY,M,DNI,T:INTEGER;
 0  0                  PARITY,IEN,IO,OEB,EA,SUM,COUT,OVR,GN,WM,Z,IND:INTEGER;
 0  0                  ADAT:TEXT;
 0  0                  AOUT:TEXT;
 0  0                  PROCEDURE  BINTODEC(I:BARRAY;N:INTEGER;VAR SUM:INTEGER);
 1  0             VAR
                     J,P:INTEGER;
 1  0             BEGIN
 1  1                  J:=1;SUM:=0;P:=1;
 1  1                  REPEAT
 1  2                        IF J=1 THEN SUM:=SUM+I[J]
 1  2                        ELSE
 1  2                            BEGIN
 1  3                                 I[J]:=I[J]*2*P;
 1  3                                 P:=2*P;
 1  3                                 SUM:=SUM+I[J]
                                 END;
 1  2                        J:=J+1;
 1  2                  UNTIL J=N+1;
 1  1              END{OF BINTODEC};
 0  0             PROCEDURE  RAMREAD(ADDRA,ADDRB:BARRAY;VAR DOUTA,DOUTB:BARRAY);
 1  0             VAR
                      I,K,J,N:INTEGER;
 1  0             BEGIN
 1  1                  N:=4;
 1  1                  BINTODEC(ADDRA,N,I);  I:=I+1;
 1  1                  BINTODEC(ADDRB,N,K);  K:=K+1;
 1  1                  FOR J:=1 TO 16 DO
 1  1                   BEGIN
 1  2                       DOUTA[J]:=RAMDATA[I,J];
 1  2                       DOUTB[J]:=RAMDATA[K,J];
 1  2                   END;
 1  1             END{RAMREAD};
 0  0             PROCEDURE RAMWRITE(ADDRB,DOUT:BARRAY;VAR RAMDATA:MAT);
 1  0             VAR
                      K,J,N:INTEGER;
 1  0             BEGIN
 1  1                  N:=4;
 1  1                  BINTODEC(ADDRB,N,K);  K:=K+1;
 1  1                  FOR J:=1 TO 16 DO RAMDATA[K,J]:=DOUT[J];
 1  1             END{RAMWRITE};
 0  0             procedure  TWOCOMP(DL:BARRAY;var BL:BARRAY);
```

```
NESTING         SOURCE TEXT   F3 ALUSIM
 1  0           var
                   J integer;
 1  0              ONEFOUND integer,
 1  0           begin
 1  1               ONEFOUND =0;
 1  1               J =16;
 1  1               REPEAT
 1  2                     if(DL[J]=1) or (ONEFOUND=1) then
 1  2                     begin
 1  3                          DL[J-1] =1-DL[J-1];
 1  3                          ONEFOUND =1;
 1  3                     end
 1  2                     else
 1  2                         DL[J] =0;
 1  2                     J =J-1;
 1  2               until J=1;
 1  1               for J =1to 16 do BL[J] =DL[J];
 1  1           end(TWOCOMP);
 0  0           procedure  ONECOMP(DL BARRAY; var BL BARRAY);
 1  0           var
                   J integer;
 1  0           begin
 1  1               J =16;
 1  1               repeat
 1  2                     DL[J] =1-DL[J];
 1  2                     J =J-1;
 1  2               until J=0;
 1  1               for J =1 to 16 do BL[J] =DL[J];
 1  1           end(ONECOMP);
 0  0           function  XOR1(A,B boolean) boolean;
 1  0           begin
 1  1               XOR1 =((NOT A AND B) OR (A AND (NOT B)));
 1  1           end(XOR1);
 0  0           procedure  MOD2ADD(A,B BARRAY;CIN integer;var C BARRAY;var COUT1 int
 1  0                                   var OVR integer);
 1  0           var
                   I,GN integer;
 1  0             SR,SS,SF,SO,SRC,SSC,SFC,CO boolean;
 1  0           begin
 1  1               for I =16 downto 1 do
 1  1               begin
 1  2                    C[I] =A[I]+B[I]+CIN;
 1  2                    CIN =0;
 1  2                    if C[I]=3 then
 1  2                    begin
 1  3                         C[I] =1;
 1  3                         CIN =1;
 1  3                    end;
 1  2                    IF C[I]=2 THEN
 1  2                    BEGIN
 1  3                         C[I] =0;
 1  3                       CIN =1;
 1  3                    END;
 1  2                       COUT1 =CIN;
 1  2               end;
```

```
NESTING         SOURCE TEXT: :F3:ALUSIM
 1   1               IF (A[1]=1) THEN SR:=TRUE ELSE SR:=FALSE;
 1   1               IF (B[1]=1) THEN SS:=TRUE ELSE SS:=FALSE;
 1   1               IF (C[1]=1) THEN SF:=TRUE ELSE SF:=FALSE;
 1   1               SO:=(SR AND SF) OR (SS AND SF) OR (SR AND SS);
 1   1               IF (SO=TRUE) THEN GN:=1 ELSE GN:=0;
 1   1               IF (SR=TRUE) THEN SRC:=FALSE ELSE SRC:=TRUE;
 1   1               IF (SS=TRUE) THEN SSC:=FALSE ELSE SSC:=TRUE;
 1   1               IF (SF=TRUE) THEN SFC:=FALSE ELSE SFC:=TRUE;
 1   1               CO:=(SRC AND SSC AND SF) OR (SR AND SS AND SFC);
 1   1               IF (CO=TRUE) THEN OVR:=1 ELSE OVR:=0;
 1   1               C[1]:=GN;
 1   1          end{MOD2ADD};
 0   0          procedure  ANDOP(A,B:BOOL;var C:BOOL);
 1   0          var
                   I:integer;
 1   0          begin
 1   1             I:=1;
 1   1             repeat
 1   2                   C[I]:=A[I] AND B[I];
 1   2                   I:=I+1;
 1   2             until I=17;
 1   1          end{ANDOP};
 0   0          procedure  XOR(A,B:BOOL;var C:BOOL);
 1   0          var
                   I:integer;
 1   0             begin
 1   1                  I:=1;
 1   1                  repeat
 1   2                        C[I]:=((NOT A[I] AND B[I]) OR (A[I] AND (NOT B[I])
 1   2                        I:=I+1;
 1   2                  until I=17;
 1   1             end{xor};
 0   0          procedure  OROP(A,B:BOOL;var C:BOOL);
 1   0          var
                   I:integer;
 1   0             begin
 1   1                  I:=1;
 1   1                  repeat
 1   2                        C[I]:=A[I] OR B[I];
 1   2                        I:=I+1;
 1   2                  until I=17;
 1   1             end{OROP};
 0   0          PROCEDURE LLS(RI:INTEGER;INP:BARRAY;VAR OP:BARRAY;VAR LO:INTEGER
 1   0          VAR
                    I:INTEGER;
 1   0          BEGIN
 1   1               OP[16]:=RI;
 1   1               I:=15;
 1   1               REPEAT
 1   2                     OP[I]:=INP[I+1];
 1   2                     I:=I-1;
 1   2               UNTIL I=0;
 1   1               LO:=INP[1];
 1   1          END{LLS};
 0   0          PROCEDURE  ALS(RI:INTEGER;INP:BARRAY;VAR OP:BARRAY;VAR LO:INTEGE
LINE NESTING        SOURCE TEXT   F3 ALUSIM
 162  1  0          VAR
                        I INTEGER;
 164  1  0          BEGIN
 165  1  1               OP[16] =RI;
 166  1  1               I =15;
 167  1  1               REPEAT
 168  1  2                     OP[I] =INP[I+1];
 169  1  2                     I =I-1;
 170  1  2               UNTIL I=1;
 171  1  1               OP[1] =INP[1];
 172  1  1               LO =INP[2];
 173  1  1          END(ALS);
 174  0  0          PROCEDURE  LRS(LI INTEGER; INP BARRAY; VAR OP BARRAY; VAR RO INTEGEI
 175  1  0          VAR
                        I INTEGER;
 177  1  0          BEGIN
 178  1  1               I.=1;
 179  1  1               OP[I] =LI;
 180  1  1               REPEAT
 181  1  2                     OP[I+1] =INP[I],
 182  1  2                     I =I+1;
 183  1  2               UNTIL I=16;
 184  1  1               RO =INP[I];
 185  1  1          END(LRS);
 186  0  0          PROCEDURE  ARS(LI INTEGER; INP BARRAY; VAR OP BARRAY; VAR RO INTEGEI
 187  1  0          VAR
                        I INTEGER;
 189  1  0          BEGIN
 190  1  1               I =1;
 191  1  1               OP[I] =INP[I];
 192  1  1               OP[I+1]· =LI;
 193  1  1               REPEAT
 194  1  2                     OP[I+2] =INP[I+1];
 195  1  2                     I =I+1;
 196  1  2               UNTIL I=15;
 197  1  1               RO =INP[I+1];
 198  1  1          END(ARS);
 199  0  0          PROCEDURE  SCFFO(IENL, DOT, A, B BOOLEAN; SFI INTEGER; VAR Z INTEGER)
 200  1  0          VAR
                        TMP1, IENN, SFN, DIN, SF: BOOLEAN;
 202  1  0          BEGIN
 203  1  1               IF XOR1(A, B)=TRUE THEN TMP1 =FALSE ELSE TMP1 =TRUE;
 204  1  1               IF (IENL=TRUE) THEN IENN =FALSE ELSE IENN =TRUE;
 205  1  1               IF (SFI=10) OR (SFI=12) THEN SF =TRUE ELSE SF =FALSE;
 206  1  1               IF (SF=TRUE) THEN SFN:=FALSE ELSE SFN =TRUE;
 207  1  1               DIN =(TMP1 AND IENN AND SF) OR (SFN AND DOT) OR (IENL AND
 208  1  1               DOT =DIN;
 209  1  1               IF (DOT=TRUE) THEN Z· =1 ELSE Z =0;
 210  1  1          END(OF SCFFO);
 211  0  0          PROCEDURE  OPSR(VAR R, S, TEMP BARRAY);
 212  1  0          VAR
                         N, J, SADDRD· INTEGER;
 214  1  0               DOUTA, DOUTB· BARRAY;
 215  1  0          BEGIN
 216  1  1               N =3;
 LINE NESTING         SOURCE TEXT: :F3:ALUSIM
  217   1   1                BINTODEC(SADDR,N,SADDRD);
  218   1   1                CASE SADDRD OF
  219   1   2             0:
                             BEGIN
  221   1   3                     RAMREAD(ADDRA,ADDRB,DOUTA,DOUTB);
  222   1   3                     FOR J:=1 TO 16 DO
  223   1   3                      BEGIN
  224   1   4                           R[J]:=DOUTA[J];
  225   1   4                           S[J]:=DOUTB[J];
  226   1   4                           TEMP[J]:=DOUTB[J];
  227   1   4                      END;
  228   1   3                END(OF 0);
  229   1   2             1:
                             BEGIN
  231   1   3                     RAMREAD(ADDRA,ADDRB,DOUTA,DOUTB);
  232   1   3                     FOR J:=1 TO 16 DO R[J]:=DOUTA[J];
  233   1   3                     FOR J:=1 TO 16 DO S[J]:=DB[J];
  234   1   3                     FOR J:=1 TO 16 DO TEMP[J]:=0;
  235   1   3                END(OF 1);
  236   1   2             2:
                             BEGIN
  238   1   3                     RAMREAD(ADDRA,ADDRB,DOUTA,DOUTB);
  239   1   3                     FOR J:=1 TO 16 DO R[J]:=DOUTA[J];
  240   1   3                     FOR J:=1 TO 16 DO S[J]:=Q[J];
  241   1   3                     FOR J:=1 TO 16 DO TEMP[J]:=DOUTB[J];
  242   1   3                END(OF 3);
  243   1   2             3:
                             BEGIN
  245   1   3                     RAMREAD(ADDRA,ADDRB,DOUTA,DOUTB);
  246   1   3                     FOR J:=1 TO 16 DO R[J]:=DOUTA[J];
  247   1   3                     FOR J:=1 TO 16 DO S[J]:=Q[J];
  248   1   3                     FOR J:=1 TO 16 DO TEMP[J]:=0;
  249   1   3                END(OF 3);
  250   1   2             4:
                             BEGIN
  252   1   3                     RAMREAD(ADDRA,ADDRB,DOUTA,DOUTB);
  253   1   3                     FOR J:=1 TO 16 DO R[J]:=DA[J];
  254   1   3                     FOR J:=1 TO 16 DO S[J]:=DOUTB[J];
  255   1   3                     FOR J:=1 TO 16 DO TEMP[J]:=DOUTB[J];
  256   1   3                END (OF 4);
  257   1   2             5:
                             BEGIN
  259   1   3                     FOR J:=1 TO 16 DO R[J]:=DA[J];
  260   1   3                     FOR J:=1 TO 16 DO S[J]:=DB[J];
  261   1   3                     FOR J:=1 TO 16 DO TEMP[J]:=0;
  262   1   3                END (OF 5);
  263   1   2             6:
                             BEGIN
  265   1   3                     RAMREAD(ADDRA,ADDRB,DOUTA,DOUTB);
  266   1   3                     FOR J:=1 TO 16 DO R[J]:=DA[J];
  267   1   3                     FOR J:=1 TO 16 DO S[J]:=Q[J];
  268   1   3                     FOR J:=1 TO 16 DO TEMP[J]:=DOUTB[J];
  269   1   3                END (OF 6);
  270   1   2             7:
                             BEGIN
```

```
      NESTING         SOURCE TEXT: :F3:ALUSIM
        1   3                   FOR J:=1 TO 16 DO R[J]:=DA[J];
        1   3                   FOR J:=1 TO 16 DO S[J]:=Q[J];
        1   3                   FOR J:=1 TO 16 DO TEMP[J]:=0;
        1   3              END {OF 7};
        1   2           END {OF CASE};
        1   1           WRITELN(SADDRD);
        1   1           FOR J:=1 TO 16 DO WRITE(R[J]);
        1   1           WRITELN;
        1   1           FOR J:=1 TO  16 DO WRITE(S[J]);
        1   1           WRITELN;
        1   1           FOR J:=1 TO 16 DO WRITE(DA[J]);
        1   1           WRITELN;
        1   1           FOR J:=1 TO 16 DO WRITE(DB[J]);
        1   1           WRITELN;
        1   1           FOR J:= 1 TO 16 DO WRITE(Q[J]);
        1   1           WRITELN;
        1   1           FOR J:=1 TO 16 DO WRITE(DOUTA[J]);
        1   1           WRITELN;
        1   1           FOR J:=1 TO 16 DO WRITE(DOUTB[J]);
        1   1           WRITELN;
        1   1          END;{OF OPSR}
        0   0         PROCEDURE  AFUN(VAR F:BARRAY;VAR COUT,OVR,GN:INTEGER);
        1   0         var
                          ONE,ONEC,RC,T1,SC,FI:BARRAY;
        1   0             INSD,N,J,OVR1,COUT1:integer;
        1   0             RCB,SB,FB,RB:BOOL;
        1   0         BEGIN
        1   1               N:=4;
        1   1               BINTODEC(I,N,INSD);
        1   1               CASE INSD OF
        1   2               0:
                              BEGIN
        1   3                      IF I[9]=1 THEN
        1   3                      BEGIN
        1   4                           FOR J:=1 TO 16 DO F[J]:=1;
        1   4                      END;
        1   3                      COUT:=0;OVR:=0;GN:=F[1];
        1   3                 END{OF 0};
        1   2               1:
                              BEGIN
        1   3                      TWOCOMP(R,RC);
        1   3                      FOR J:=1 TO 15 DO ONE[J]:=0;ONE[16]:=1;
        1   3                      TWOCOMP(ONE,ONEC);
        1   3                      MOD2ADD(S,RC,CIN,T1,COUT1,OVR1);
        1   3                      MOD2ADD(T1,ONEC,OVR1,F,COUT,OVR);
        1   3                      GN:=F[1];
        1   3                 END{OF 1};
        1   2               2:
                              BEGIN
        1   3                      TWOCOMP(S,SC);
        1   3                      FOR J:=1 TO 15 DO ONE[J]:=0;  ONE[16]:=1;
        1   3                      TWOCOMP(ONE,ONEC);
        1   3                      MOD2ADD(R,SC,CIN,T1,COUT1,OVR1);
        1   3                      MOD2ADD(T1,ONEC,OVR1,F,COUT,OVR);
        1   3                      GN:=F[1];
```

```
 LINE NESTING         SOURCE TEXT: :F3:ALUSIM
        1   3                    END(OF 2);
        1   2                 3:
                                 BEGIN
        1   3                           MOD2ADD(R,S,CIN,F,COUT,OVR);
        1   3                           GN:=F[1];
        1   3                    END(OF 3);
        1   2                 4:
                                 BEGIN
        1   3                           FOR J:=1 TO 16 DO R[J]:=0;
        1   3                           MOD2ADD(S,R,CIN,F,COUT,OVR);
        1   3                           GN:=F[1];
        1   3                    END(OF 4);
        1   2                 5:
                                 BEGIN
        1   3                           FOR J:=1 TO 16 DO R[J]:=0;
        1   3                           ONECOMP(S,SC);
        1   3                           MOD2ADD(SC,R,CIN,F,COUT,OVR);
        1   3                           GN:=F[1];
        1   3                    END(OF 5);
        1   2                 6:
                                 BEGIN
        1   3                           FOR J:=1 TO 16 DO S[J]:=0;
        1   3                           MOD2ADD(R,S,CIN,F,COUT,OVR);
        1   3                           GN:=F[1];
        1   3                    END(OF 6);
        1   2                 7:
                                 BEGIN
        1   3                           FOR J:=1 TO 16 DO S[J]:=0;
        1   3                           ONECOMP(R,RC);
        1   3                           MOD2ADD(RC,S,CIN,F,COUT,OVR);
        1   3                           GN:=F[1];
        1   3                    END(OF 7);
        1   2                 8:
                                 BEGIN FOR J:=1 TO 16 DO F[J]:=0;
        1   3                    COUT:=0;OVR:=0;GN:=0;
        1   3                    END(OF 8);
        1   2                 9:
                                 BEGIN
        1   3                           ONECOMP(R,RC);
        1   3                           FOR J:=1 TO 16 DO
        1   3                           BEGIN
        1   4                                IF RC[J]=1 THEN RCB[J]:=TRUE
        1   4                                ELSE RCB[J]:=FALSE;
        1   4                           END;
        1   3                           FOR J:=1 TO 16 DO
        1   3                           BEGIN
        1   4                                IF S[J]=1 THEN SB[J]:=TRUE
        1   4                                ELSE SB[J]:=FALSE;
        1   4                           END;
        1   3                           ANDOP(RCB,SB,FB);
        1   3                           COUT:=0; OVR:=0;
        1   3                           FOR J:=1 TO 16 DO
        1   3                           BEGIN
        1   4                                IF FB[J]=TRUE THEN F[J]:=1
        1   4                                ELSE F[J]:=0;
```

```
LINE NESTING        SOURCE TEXT:  :F3:ALUSIM
 382   1   4                      END;
 383   1   3                      GN:=F[1];
 384   1   3                 END{OF 9};
 385   1   2              10:
                             BEGIN
 387   1   3                      FOR J:=1 TO 16 DO
 388   1   3                      BEGIN
 389   1   4                           IF R[J]=1 THEN RB[J]:=TRUE
 390   1   4                           ELSE RB[J]:=FALSE;
 391   1   4                      END;
 392   1   3                      FOR J:=1 TO 16 DO
 393   1   3                      BEGIN
 394   1   4                           IF S[J]=1 THEN SB[J]:=TRUE
 395   1   4                           ELSE SB[J]:=FALSE;
 396   1   4                      END;
 397   1   3                      XOR(RB,SB,FB);
 398   1   3                      FOR J:=1 TO 16 DO
 399   1   3                      BEGIN
 400   1   4                           IF FB[J]=TRUE THEN FI[J]:=1 ELSE FI[J]:=0;
 401   1   4                      END;
 402   1   3                      ONECOMP(FI,F);
 403   1   3                      COUT:=0; OVR:=0; GN:=F[1];
 404   1   3                 END {OF10};
 405   1   2              11:
                             BEGIN
 407   1   3                      FOR J:=1 TO 16 DO
 408   1   3                      BEGIN
 409   1   4                           IF R[J]=1 THEN RB[J]:=TRUE
 410   1   4                           ELSE RB[J]:=FALSE;
 411   1   4                      END;
 412   1   3                      FOR J:=1 TO 16 DO
 413   1   3                      BEGIN
 414   1   4                           IF S[J]=1 THEN SB[J]:=TRUE
 415   1   4                           ELSE SB[J]:=FALSE;
 416   1   4                      END;
 417   1   3                      XOR(RB,SB,FB);
 418   1   3                      FOR J:=1 TO 16 DO
 419   1   3                      BEGIN
 420   1   4                           IF FB[J]=TRUE THEN F[J]:=1
 421   1   4                           ELSE F[J]:=0;
 422   1   4                      END;
 423   1   3                      COUT:=0; OVR:=0; GN:=F[1];
 424   1   3                 END{OF 11};
 425   1   2              12:
                             BEGIN
 427   1   3                      FOR J:=1 TO 16 DO
 428   1   3                      BEGIN
 429   1   4                           IF R[J]=1 THEN RB[J]:=TRUE
 430   1   4                           ELSE RB[J]:=FALSE;
 431   1   4                      END;
 432   1   3                      FOR J:=1 TO 16 DO
 433   1   3                      BEGIN
 434   1   4                           IF S[J]=1 THEN SB[J]:=TRUE
 435   1   4                           ELSE SB[J]:=FALSE;
 436   1   4                      END;
```

```
LINE NESTING        SOURCE TEXT: :F3:ALUSIM
                ANDOP(RB,SB,FB);
                FOR J:=1 TO 16 DO
                BEGIN
                     IF FB[J]=TRUE THEN F[J]:=1
                     ELSE F[J]:=0;
                END;
                COUT:=0; OVR:=0; GN:=F[1];
           END(OF 12);
        13:
           BEGIN
                FOR J:=1 TO 16 DO
                BEGIN
                     IF R[J]=1 THEN RB[J]:=TRUE
                     ELSE RB[J]:=FALSE;
                END;
                FOR J:=1 TO 16 DO
                BEGIN
                     IF S[J]=1 THEN SB[J]:=TRUE
                     ELSE SB[J]:=FALSE;
                END;
                OROP(RB,SB,FB);
                FOR J:=1 TO 16 DO
                BEGIN
                     IF FB[J]=TRUE THEN FI[J]:=1
                     ELSE FI[J]:=0;
                END;
                ONECOMP(FI,F);
                COUT:=0; OVR:=0; GN:=F[1];
           END(OF 13);
        14:
           BEGIN
                FOR J:=1 TO 16 DO
                BEGIN
                     IF R[J]=1 THEN RB[J]:=TRUE
                     ELSE RB[J]:=FALSE;
                END;
                FOR J:=1 TO 16 DO
                BEGIN
                     IF S[J]=1 THEN SB[J]:=TRUE
                     ELSE SB[J]:=FALSE;
                END;
                ANDOP(RB,SB,FB);
                FOR J:=1 TO 16 DO
                BEGIN
                     IF FB[J]=TRUE THEN FI[J]:=1
                     ELSE FI[J]:=0;
                END;
                ONECOMP(FI,F);
                COUT:=0; OVR:=0; GN:=F[1];
           END(OF 14);
        15:
           BEGIN
                FOR J:=1 TO 16 DO
                BEGIN
                     IF R[J]=1 THEN RB[J]:=TRUE
```

```
NESTING          SOURCE TEXT: :F3:ALUSIM
 1  4                          ELSE RB[J]:=FALSE;
 1  4                      END;
 1  3                      FOR J:=1 TO 16 DO
 1  3                      BEGIN
 1  4                          IF S[J]=1 THEN SB[J]:=TRUE
 1  4                         ELSE SB[J]:=FALSE;
 1  4                      END;
 1  3                      OROP(RB,SB,FB);
 1  3                      FOR J:=1 TO 16 DO
 1  3                      BEGIN
 1  4                          IF FB[J]=TRUE THEN F[J]:=1
 1  4                          ELSE F[J]:=0;
 1  4                      END;
 1  3                      COUT:=0;  OVR:=0;  GN:=F[1];
 1  3                  END(OF 15);
 1  2                END(OF CASE);
 1  1                WRITELN('INSD=',INSD);
 1  1                FOR J:=1 TO 16 DO WRITE(R[J]);
 1  1                WRITELN;
 1  1                FOR J:=1 TO 16 DO WRITE(S[J]);
 1  1                WRITELN;
 1  1                FOR J:=1 TO 16 DO WRITE(F[J]);
 1  1                WRITELN;
 1  1                WRITE('COUT=',COUT:3,'    OVR=',OVR:3,'    GN=',GN:3);
 1  1                WRITELN;
 1  1            end;(AFUN)
 0  0            PROCEDURE   DSTF(VAR ASO,Y,Q,QRIN:BARRAY;VAR SIO3,SIO0,QIO3,
 1  0                                     PARITY,Z:INTEGER);
 1  0            VAR
                    J,S1,S2,S3,INSD,N:INTEGER;
 1  0            BEGIN
 1  1                FOR J:=1 TO 4 DO INT[J]:=I[J+4];
 1  1                N:=4;
 1  1                BINTODEC(INT,N,INSD);
 1  1                CASE INSD OF
 1  2                           0:
                                  BEGIN
 1  3                             ARS(SIO3,F,ASO,SIO0);
 1  3                             WM:=0;
 1  3                             QIO3:=-1;
 1  3                             QIO0:=-1;
 1  3                             END(OF ARS);
 1  2                           1:
                                  BEGIN
 1  3                                 LRS(SIO3,F,ASO,SIO0);
 1  3                                 WM:=0;
 1  3                                 QIO3:=-1;
 1  3                                 QIO0:=-1;
 1  3                             END(OF LRS);
 1  2                           2:
                                  BEGIN
 1  3                                 ARS(SIO3,F,ASO,SIO0);
 1  3                                 WM:=0;
 1  3                                 LRS(QIO3,Q,QRIN,QIO0);
 1  3                             END(OF ALRS);
```

```
NESTING        SOURCE TEXT: :F3:ALUSIM
 1  2                     3:
                            BEGIN
 1  3                           LRS(SIO3,F,ASO,SIOO);
 1  3                           WM:=0;
 1  3                           LRS(QIO3,Q,QRIN,QIOO);
 1  3                       END(OF LLRS);
 1  2                     4:
                            BEGIN
 1  3                           FOR J:=1 TO 16 DO ASO[J]:=F[J];
 1  3                           SIO3:=INP;
 1  3                           WM:=0;
 1  3                           QIO3:=-1;
 1  3                           QIOO:=-1;
 1  3                           S1:=0;
 1  3                           FOR J:=1 TO 16 DO S1:=F[J]+S1;
 1  3                           S2:=S1+SIO3;
 1  3                           S3:=S2 MOD 2;
 1  3                           IF S3=0 THEN PARITY:=0 ELSE PARITY:=1;
 1  3                           SIOO:=PARITY;
 1  3                       END(OF 4);
 1  2                     5:
                            BEGIN
 1  3                           FOR J:=1 TO 16 DO ASO[J]:=F[J];
 1  3                           SIO3:=INP;
 1  3                           WM:=1;
 1  3                           LRS(QIO3,Q,QRIN,QIOO);
 1  3                           S1:=0;
 1  3                           FOR J:=1 TO 16 DO S1:=F[J]+S1;
 1  3                           S2:=S1+SIO3;
 1  3                           S3:=S2 MOD 2;
 1  3                           IF S3=0 THEN PARITY:=0 ELSE PARITY:=1;
 1  3                           SIOO:=PARITY;
 1  3                       END(OF 5);
 1  2                     6:
                            BEGIN
 1  3                           FOR J:=1 TO 16 DO ASO[J]:=F[J];
 1  3                           SIO3:=INP;
 1  3                           WM:=1;
 1  3                           FOR J:= 1 TO 16 DO QRIN[J]:=F[J];
 1  3                           QIO3:=-1;
 1  3                           QIOO:=-1;
 1  3                           S1:=0;
 1  3                           FOR J:=1 TO 16 DO S1:=F[J]+S1;
 1  3                           S2:=S1+SIO3;
 1  3                           S3:=S2 MOD 2;
 1  3                           IF S3=0 THEN PARITY:=0 ELSE PARITY:=1;
 1  3                           SIOO:=PARITY;
 1  3                       END(OF 6);
 1  2                     7:
                            BEGIN
 1  3                           FOR J:=1 TO 16 DO ASO[J]:=F[J];
 1  3                           SIO3:=INP;
 1  3                           WM:=0;
 1  3                           FOR J:=1 TO 16 DO QRIN[J]:=F[J];
 1  3                           QIO3:=-1;
```

```
SOURCE TEXT: :F3:ALUSIM
                QIOO:=-1;
                S1:=0;
                FOR J:=1 TO 16 DO S1:=F[J]+S1;
                S2:=S1+SIO3;
                S3:=S2 MOD 2;
                IF S3=0 THEN PARITY:=0 ELSE PARITY:=1;
                SIOO:=PARITY;
          END(OF 7);
      8:
          BEGIN
                ALS(SIOO,F,ASO,SIO3);
                WM:=0;
                QIO3:=-1;
                QIOO:=-1;
          END(OF ALS);
      9:
          BEGIN
                LLS(SIOO,F,ASO,SIO3);
                WM:=0;
                QIO3:=-1;
                QIOO:=-1;
          END(OF LLS);
      10:
          BEGIN
                ALS(SIOO,F,ASO,SIO3);
                WM:=0;
                LLS(QIOO,Q,QRIN,QIO3);
          END(OF ALLS);
      11:
          BEGIN
                LLS(SIOO,F,ASO,SIO3);
                WM:=0;
                LLS(QIOO,Q,QRIN,QIO3);
          END(OF LLLS);
      12:
          BEGIN
                FOR J:=1 TO 16 DO ASO[J]:=F[J];
                SIO3:=F[1];
                SIOO:=-1;
                WM:=1;
                QIO3:=-1;
                QIOO:=-1;
          END(OF 12);
      13:
          BEGIN
                FOR J:=1 TO 16 DO ASO[J]:=F[J];
                SIO3:=F[1];
                SIOO:=-1;
                WM:=1;
                LLS(QIOO,Q,QRIN,QIO3);
          END(OF LLS);
      14:
          BEGIN
                SIOO:=INP;
                FOR J:=1 TO 16 DO ASO[J]:=SIOO;
```

```
NESTING        SOURCE TEXT   F3 ALUSIM
 1  3                                 SI03 =SI00;
 1  3                                 WM =0,
 1  3                                 QI03 =-1;
 1  3                                 QI00 =-1;
 1  3                            END(OF 14);
 1  2                         15
                                 BEGIN
 1  3                                 FOR J =1 TO 16 DO ASO[J] =F[J];
 1  3                                 SI03 =F[1];
 1  3                                 SI00 =-1;
 1  3                                 WM =0;
 1  3                                 QI03 =-1;
 1  3                                 QI00 =-1;
 1  3                            END(OF 15);
 1  2                       END(OF CASE);
 1  1                       IF (OEY=0) THEN
 1  1                         BEGIN
 1  2                              FOR J =1 TO 16 DO Y[J] =ASO[J];
 1  2                              S1 =0;
 1  2                              FOR J =1 TO 16 DO S1 =Y[J]+S1;
 1  2                              IF S1=0 THEN Z =1 ELSE Z =0;
 1  2                         END
 1  1                         ELSE
 1  1                             BEGIN
 1  2                                   Z =9;
 1  2                                   FOR J =1 TO 16 DO Y[J] =OD[J],
 1  2                             END;
 1  1                       WRITELN(INSD);
 1  1                       FOR J =1 TO 16 DO WRITE(F[J]);
 1  1                       WRITELN;
 1  1                       WRITE(SI03,SI00);
 1  1                       WRITELN;
 1  1                       WRITE(QI03,QI00);
 1  1                       WRITELN;
 1  1                       FOR J =1 TO 16 DO WRITE(Y[J]);
 1  1                       WRITELN;
 1  1                       FOR J =1 TO 16 DO WRITE(Q[J]);
 1  1                       WRITELN;
 1  1                       FOR J =1 TO 16 DO WRITE(QRIN[J]);
 1  1                       WRITELN;
 1  1                       WRITE(WM,Z);
 1  1                       WRITELN;
 1  1               END(OF DSTF);
 0  0          PROCEDURE  SPLF(I,R,S BARRAY;CIN,INP INTEGER;VAR F,ASO,QRIN,Y BARRA
 1  0                            VAR SI03,SI00,QI03,QI00,COUT,OVR,GN,WM,Z INTEGER);
 1  0          VAR
                    SC,RC,ONE,ONEC,INT,T1 BARRAY;
 1  0               INSD,J,N,S1,S2,S3,PARITY,COUT1,OVR1 INTEGER;
 1  0               IENL,R1L,F1L,DOT,FL,OVRL,ASOL,SL BOOLEAN;
 1  0          BEGIN
 1  1               FOR J:=1 TO 4 DO INT[J]:=I[J+4];
 1  1               N:=4;
 1  1               BINTODEC(INT,N,INSD);
 1  1               CASE INSD OF
 1  2             0
```

```
LINE NESTING        SOURCE TEXT: :F3:ALUSIM
   BEGIN
        IF (OEY=0) THEN Z:=Q[16];
        IF (Z=0) THEN
          BEGIN
               FOR J:=1 TO 16 DO R[J]:=0;
               MOD2ADD(S,R,CIN,F,COUT,OVR);
          END
          ELSE MOD2ADD(R,S,CIN,F,COUT,OVR);
        GN:=F[1]; SIO3:=Z;
        LRS(SIO3,F,ASO,SIO0);
        ASO[1]:=COUT;
        LRS(QIO3,Q,QRIN,QIO0);
   END(OF UM);
1,3,7,9,11,13,15:
                 BEGIN
                      WRITE('NOT IMPLEMENTED IN AM2903');
                      WRITELN;
                 END;
2:
   BEGIN
        IF (OEY=0) THEN Z:=Q[16];
        IF (Z=0) THEN
          BEGIN
               FOR J:=1 TO 16 DO R[J]:=0;
               MOD2ADD(S,R,CIN,F,COUT,OVR);
          END
          ELSE MOD2ADD(R,S,CIN,F,COUT,OVR);
        GN:=F[1]; SIO3:=Z;
        LRS(SIO3,F,ASO,SIO0);
        IF (F[1]=1) THEN FL:=TRUE ELSE FL:=FALSE;
        IF (OVR=1) THEN OVRL:=TRUE ELSE OVRL:=FALSE;
        ASOL:=XOR1(FL,OVRL);
        IF (ASOL=TRUE) THEN ASO[1]:=1 ELSE ASO[1]:=0;
        LRS(QIO3,Q,QRIN,QIO0);
   END(OF TCM);
4:
   BEGIN
        FOR J:=1 TO 15 DO R[J]:=0;
        R[16]:=1;
        MOD2ADD(S,R,CIN,F,COUT,OVR);
        GN:=F[1];
        FOR J:=1 TO 16 DO ASO[J]:=F[J];
        SIO3:=INP;
        S1:=0;
        FOR J:=1 TO 16 DO S1:=F[J]+S1;
        S2:=S1+SIO3;
        S3:=S2 MOD 2;
        IF (S3=0) THEN PARITY:=0 ELSE PARITY:=1;
        SIO0:=PARITY;
        IF (OEY=0) THEN
          BEGIN
               FOR J:=1 TO 16 DO Y[J]:=ASO[J];
               S1:=0;
               FOR J:=1 TO 16 DO S1:=Y[J]+S1;
               IF S1=0 THEN Z:=1 ELSE Z:=0;
```

```
LINE NESTING        SOURCE TEXT: :F3:ALUSIM
 767   1   4                   END;
 768   1   3                 QIO3:=Z;
 769   1   3                 QIO0:=Z;
 770   1   3            END(OF IO0/R2);
 771   1   2         5:
                        BEGIN
 773   1   3                 IF (OEY=0) THEN Z:=S[1];
 774   1   3                 IF (Z=0) THEN
 775   1   3                   BEGIN
 776   1   4                        FOR J:=1 TO 16 DO R[J]:=0;
 777   1   4                        MOD2ADD(S,R,CIN,F,COUT,OVR);
 778   1   4                        GN:=F[1];
 779   1   4                   END
 780   1   3                 ELSE
 781   1   3                     BEGIN
 782   1   4                        FOR J:=1 TO 16 DO R[J]:=0;
 783   1   4                        ONECOMP(S,SC);
 784   1   4                        MOD2ADD(SC,R,CIN,F,COUT,OVR);
 785   1   4                        S1:=F[1]+S[1];
 786   1   4                        IF (S1=1) THEN GN:=1 ELSE GN:=0;
 787   1   4                     END;
 788   1   3                 FOR J:=1 TO 16 DO ASO[J]:=F[J];
 789   1   3                 IF (S[1]=1) THEN SL:=TRUE ELSE SL:=FALSE;
 790   1   3                 IF (F[1]=1) THEN FL:=TRUE ELSE FL:=FALSE;
 791   1   3                 ASOL:=XOR1(SL,FL);
 792   1   3                 IF (ASOL=TRUE) THEN ASO[1]:=1 ELSE ASO[1]:=0;
 793   1   3                 S1:=S[1]+F[1];
 794   1   3                 IF(S1=1) THEN ASO[1]:=1 ELSE ASO[1]:=0;
 795   1   3                 SIO3:=INP;
 796   1   3                 S1:=0;
 797   1   3                 FOR J:=1 TO 16 DO S1:=F[J]+S1;
 798   1   3                 S2:=S1+SIO3;
 799   1   3                 S3:=S2 MOD 2;
 800   1   3                 IF (S3=0) THEN PARITY:=0 ELSE PARITY:=1;
 801   1   3                 SIO0:=PARITY;
 802   1   3                 QIO3:=Z;
 803   1   3                 QIO0:=Z;
 804   1   3            END(OF SMTC);
 805   1   2         6:
                        BEGIN
 807   1   3                 IF (OEY=0) THEN Z:=Q[16];
 808   1   3                 IF (Z=0) THEN
 809   1   3                   BEGIN
 810   1   4                        FOR J:=1 TO 16 DO R[J]:=0;
 811   1   4                        MOD2ADD(S,R,CIN,F,COUT,OVR);
 812   1   4                   END
 813   1   3                 ELSE
 814   1   3                      BEGIN
 815   1   4                             TWOCOMP(R,RC);
 816   1   4                             FOR J:=1 TO 15 DO ONE[J]:=0;
 817   1   4                             ONE[16]:=1;
 818   1   4                             TWOCOMP(ONE,ONEC);
 819   1   4                             MOD2ADD(S,RC,CIN,T1,COUT1,OVR1);
 820   1   4                             MOD2ADD(T1,ONEC,OVR1,F,COUT,OVR);
 821   1   4                       END;
```

```
LINE NESTING         SOURCE TEXT: :F3:ALUSIM
 822   1  3                GN:=F[1]; SIO3:=Z;
 823   1  3                LRS(SIO3,F,ASO,SIO0);
 824   1  3                IF (F[1]=1) THEN FL:=TRUE ELSE FL:=FALSE;
 825   1  3                IF (OVR=1) THEN OVRL:=TRUE ELSE OVRL:=FALSE;
 826   1  3                ASOL:=XOR1(FL,OVRL);
 827   1  3                IF (ASOL=TRUE) THEN ASO[1]:=1 ELSE ASO[1]:=0;
 828   1  3                LRS(QIO3,Q,QRIN,QIO0);
 829   1  3            END(OF TCMLC);

 831   1  2          8:
                        BEGIN
 833   1  3                FOR J:=1 TO 16 DO R[J]:=0;
 834   1  3                MOD2ADD(S,R,CIN,F,COUT,OVR);
 835   1  3                GN:=Q[1];
 836   1  3                FOR J:=1 TO 16 DO ASO[J]:=F[J];
 837   1  3                SIO3:=F[1];
 838   1  3                LLS(QIO0,Q,QRIN,QIO3);
 839   1  3                IF (OEY=0) THEN
 840   1  3                  BEGIN
 841   1  4                       S1:=0;
 842   1  4                       FOR J:=1 TO 16 DO S1:=QRIN[J]+S1;
 843   1  4                       IF (S1=0) THEN Z:=1 ELSE Z:=0;
 844   1  4                  END;
 845   1  3               SIO0:=Z;
 846   1  3         END(OF SLN);
 847   1  2          10:
                        BEGIN
 849   1  3                FOR J:=1 TO 16 DO R[J]:=0;
 850   1  3                MOD2ADD(S,R,CIN,F,COUT,OVR);
 851   1  3                LLS(SIO0,F,ASO,SIO3);
 852   1  3                LLS(QIO0,Q,QRIN,QIO3);
 853   1  3                GN:=F[1];
 854   1  3                S1:=R[1]+F[1];
 855   1  3                IF (S1=1) THEN SIO3:=1 ELSE SIO3:=0;
 856   1  3                IF (OEY=0) THEN
 857   1  3                  BEGIN
 858   1  4                       S1:=0;
 859   1  4                       FOR J:=1 TO 16 DO S1:=QRIN[J]+S1;
 860   1  4                       S2:=0;
 861   1  4                       FOR J:=1 TO 16 DO S2:=F[J]+S2;
 862   1  4                       S3:=S2+S1;
 863   1  4                       IF (S3=0) THEN Z:=1 ELSE Z:=0;
 864   1  4                  END;
 865   1  3         END(OF DLN&FDOP);
 866   1  2          12:
                        BEGIN
 868   1  3                IF (OEY=0) THEN
 869   1  3                BEGIN
 870   1  4                     IF (IEN=1) THEN IENL:=TRUE ELSE IENL:=FALSE;
 871   1  4                     IF (R[1]=1) THEN R1L:=TRUE ELSE R1L:=FALSE;
 872   1  4                     IF (F[1]=1) THEN F1L:=TRUE ELSE F1L:=FALSE;
 873   1  4                     SCFFO(IENL,DOT,R1L,F1L,INSD,Z);
 874   1  4                END;
 875   1  3                IF (Z=0) THEN MOD2ADD(S,R,CIN,F,COUT,OVR)
 876   1  3                ELSE
```

```
LINE NESTING        SOURCE TEXT: :F3:ALUSIM
877   1  3                     BEGIN
878   1  4                          TWOCOMP(R,RC);
879   1  4                          FOR J:=1 TO 15 DO ONE[J]:=0;
880   1  4                          ONE[16]:=1;
881   1  4                          TWOCOMP(ONE,ONEC);
882   1  4                          MOD2ADD(S,RC,CIN,T1,COUT1,OVR1);
883   1  4                          MOD2ADD(T1,ONEC,OVR1,F,COUT,OVR);
884   1  4                     END;
885   1  3                 GN:=F[1];
886   1  3                 LLS(SIO0,F,AS0,SIO3);
887   1  3                 LLS(QIO0,Q,QRIN,QIO3);
888   1  3                 S1:=F[1]+R[1];
889   1  3                 IF (S1=1) THEN SIO3:=0 ELSE SIO3:=1;
890   1  3            END{OF TCD};
891   1  2         14:
                      BEGIN
893   1  3                 IF (OEY=0) THEN
894   1  3                   BEGIN
895   1  4                        IF(IEN=1) THEN IENL:=TRUE ELSE IENL:=FALSE;
896   1  4                        IF (R[1]=1) THEN R1L:=TRUE ELSE R1L:=FALSE;
897   1  4                        IF (F[1]=1) THEN F1L:=TRUE ELSE F1L:=FALSE;
898   1  4                        SCFFO(IENL,DOT,R1L,F1L,INSD,Z);
899   1  4                   END;
900   1  3                 IF (Z=0) THEN MOD2ADD(S,R,CIN,F,COUT,OVR)
901   1  3                 ELSE
902   1  3                     BEGIN
903   1  4                          TWOCOMP(R,RC);
904   1  4                          FOR J:=1 TO 15 DO ONE[J]:=0;
905   1  4                          ONE[16]:=1;
906   1  4                          TWOCOMP(ONE,ONEC);
907   1  4                          MOD2ADD(S,RC,CIN,T1,COUT1,OVR1);
908   1  4                          MOD2ADD(T1,ONEC,OVR1,F,COUT,OVR);
909   1  4                     END;
910   1  3                 GN:=F[1];
911   1  3                 FOR J:=1 TO 16 DO AS0[J]:=F[J];
912   1  3                 SIO3:=F[1];
913   1  3                 SIO0:=Z;
914   1  3                 LLS(QIO0,Q,QRIN,QIO3);
915   1  3            END{OF TCDC&R};
916   1  2          END{OF CASE};
917   1  1          WM:=0;
918   1  1          IF (OEY=0) THEN
919   1  1            BEGIN
920   1  2                 FOR J:=1 TO 16 DO Y[J]:=AS0[J];
921   1  2            END
922   1  1          ELSE
923   1  1              BEGIN
924   1  2                   FOR J:=1 TO 16 DO Y[J]:=OD[J];
925   1  2              END;
926   1  1          WRITELN(INSD);
927   1  1          FOR J:=1 TO 16 DO WRITE(R[J]);
928   1  1          WRITELN;
929   1  1          FOR J:=1 TO 16 DO WRITE(S[J]);
930   1  1          WRITELN;
931   1  1          FOR J:=1 TO 16 DO WRITE(F[J]);
```

```
LINE NESTING         SOURCE TEXT: :F3:ALUSIM
 932  1  1               WRITELN;
 933  1  1               FOR J:=1 TO 16 DO WRITE(Y[J]);
 934  1  1               WRITELN;
 935  1  1               FOR J:=1 TO 16 DO WRITE(QRIN[J]);
 936  1  1               WRITELN;
 937  1  1               WRITE(COUT,OVR,GN,Z);
 938  1  1               WRITELN;
 939  1  1               WRITE(SIO3,SIO0,QIO3,QIO0);
 940  1  1               WRITELN;
 941  1  1              END(OF SPLF);
 942  0  0           BEGIN(MAIN)
 943  0  1                RESET(ADAT,':F3:ASIM.DAT');
 944  0  1                REWRITE(AOUT,':F3:SIMALU.OUT');
 945  0  1                READ(ADAT,M);
 946  0  1                FOR IND:=1 TO M DO
 947  0  1                   BEGIN
 948  0  2                        FOR J:=1 TO 4 DO READ(ADAT,ADDRB[J]);
 949  0  2                        FOR J:=1 TO 16 DO READ(ADAT,DIN[J]);
 950  0  2                        RAMWRITE(ADDRB,DIN,RAMDATA);
 951  0  2                   END;
 952  0  1                READ(ADAT,T);
 953  0  1                FOR DNI:=1 TO T DO
 954  0  1           BEGIN
 955  0  2                FOR J:=1 TO 9 DO READ(ADAT,I[J]);
 956  0  2                FOR J:=1 TO 4 DO READ(ADAT,ADDRA[J]);
 957  0  2                FOR J:=1 TO 4 DO READ(ADAT,ADDRB[J]);
 958  0  2                FOR J:=1 TO 16 DO READ(ADAT,DA[J]);
 959  0  2                FOR J:=1 TO 16 DO READ(ADAT,DB[J]);
 960  0  2                READ(ADAT,EA,OEB,CIN);
 961  0  2                READ(ADAT,SIO3,SIO0,QIO3,QIO0);
 962  0  2                READ(ADAT,IEN,OEY,INP);
 963  0  2                I0:=I[9];
 964  0  2                SADDR[1]:=OEB;
 965  0  2                SADDR[2]:=I0;
 966  0  2                SADDR[3]:=EA;
 967  0  2                OPSR(R,S,TEMP);
 968  0  2                IF (OEB=0) THEN
 969  0  2                  BEGIN
 970  0  3                       FOR J:=1 TO 16 DO DB[J]:=TEMP[J];
 971  0  3                  END;
 972  0  2                SUM:=I[1]+I[2]+I[3]+I[4]+I[9];
 973  0  2                IF (SUM<>0) THEN
 974  0  2                  BEGIN
 975  0  3                       AFUN(F,COUT,OVR,GN);
 976  0  3                       DSTF(ASO,Y,Q,QRIN,SIO3,SIO0,QIO3,QIO0,WM,PARITY,Z);
 977  0  3                  END
 978  0  2                ELSE
 979  0  2                    BEGIN
 980  0  3                         SPLF(I,R,S,CIN,INP,F,ASO,QRIN,Y,SIO3,SIO0,
                                         QIO3,QIO0,COUT,OVR,GN,WM,Z);
 982  0  3                    END;
 983  0  2                IF (IEN=0) THEN
 984  0  2                  BEGIN
 985  0  3                       FOR J:= 1 TO 16 DO Q[J]:=QRIN[J];
 986  0  3                       IF (WM=0) THEN RAMWRITE(ADDRB,Y,RAMDATA);
```

```
LINE NESTING        SOURCE TEXT: :F3:ALUSIM
 987   0  3                   END
 988   0  2                 ELSE
 989   0  2                     BEGIN
 990   0  3                          WM:=1;
 991   0  3                          FOR J:=1 TO 16 DO Q[J]:=Q[J];
 992   0  3                     END;
 993   0  2                FOR J:= 1 TO 4 DO WRITE(ADDRA[J]);
 994   0  2                WRITELN;
 995   0  2                FOR J:=1 TO 4 DO WRITE(ADDRB[J]);
 996   0  2                WRITELN;
 997   0  2                FOR J:=1 TO 9 DO WRITE(I[J]);
 998   0  2                WRITELN;
 999   0  2                FOR J:= 1 TO 16 DO WRITE(DA[J]);
1000   0  2                WRITELN;
1001   0  2                FOR J:= 1 TO 16 DO WRITE(DB[J]);
1002   0  2                WRITELN;
1003   0  2                WRITE(EA,OEB,CIN);
1004   0  2                WRITELN;
1005   0  2                WRITE(OEY,IEN,WM);
1006   0  2                WRITELN;
1007   0  2                WRITE(COUT,OVR,N,Z);
1008   0  2                WRITELN;
1009   0  2                WRITE(SIO3,SIO0,QIO3,QIO0);
1010   0  2                WRITELN;
1011   0  2                FOR J:=1 TO 16 DO WRITE(F[J]);
1012   0  2                WRITELN;
1013   0  2                FOR J:=1 TO 16 DO WRITE(ASO[J]);
1014   0  2                WRITELN;
1015   0  2                FOR J:= 1 TO 16 DO WRITE(Y[J]);
1016   0  2                WRITELN;
1017   0  2                FOR J:=1 TO 16 DO WRITE(OD[J]);
1018   0  2                WRITELN;
1019   0  2                FOR J:=1 TO 16 DO WRITE(Q[J]);
1020   0  2                WRITELN;
1021   0  2                FOR J:=1 TO 16 DO WRITE(QRIN[J]);
1022   0  2                WRITELN;
1023   0  2            END;
1024   0  1           END.
```

```
ary Information:

EDURE                OFFSET   CODE SIZE     DATA SIZE    STACK SIZE
ODEC                  005BH   0072H   114D               0008H     8D
EAD                   00CDH   0094H   148D               0038H    56D
RITE                  0161H   005CH    92D               0036H    54D
OMP                   01BDH   0075H   117D               000CH    12D
OMP                   0232H   004CH    76D               000AH    10D
                      027EH   0026H    38D               0006H     6D
ADD                   02A4H   0145H   325D               0014H    20D
P                     03E9H   002BH    43D               0006H     6D
                      0414H   0038H    56D               0006H     6D
```

```
•III Pascal-86, V2 0
                                    Summary Information


                       044CH  002BH     43D                     0006H       6D
                       0477H  0043H     67D                     0006H       6D
                       04BAH  0049H     73D                     0006H       6D
                       0503H  004DH     77D                     0006H       6D
                       0550H  0063H     99D                     0006H       6D
                       05B3H  0096H    150D                     0010H      16D
                       0649H  05BAH   1466D                     0108H     264D
                       0C03H  0C44H   3140D                     01DEH     478D
                       1847H  08DEH   2270D                     0094H     148D
                       2125H  10F5H   4341D                     01A6H     422D
                       321AH  0776H   1910D  04CEH    1230D     009AH     154D
 IN CODE-                     005BH     91D

                              3990H 14736D   04CEH    1230D     06CEH    1742D

ines Read
rrors Detected
tilization of Memory
```

```
  Pascal-86, V2.0


ile:  :F3:SEQ.SIM
ile:  :F3:SEQ.OBJ
 Specified: <none>.

E NESTING        SOURCE TEXT: :F3:SEQ.SIM
1  0  0          PROGRAM AM2910(INP,OUTPUT);
2  0  0               (* -----------------------------------------------------------
                      (*
                      (*               MICROPROGRAMME CONTROLLER
                      (*
                      (*      EXPLANATION OF VARIABLES:-
                      (*
                      (*      MAXCOUNT      :  MAXIMUM MICROWORD ADDRESSABLE RANGE
                      (*      STACKSIZE     :  DEPTH OF STACK
                      (*      STACKARRAY    :  STACK EXPRESSED IN ARRAY
                      (*      POINTERLIMITS:  RANGE OF STACK POINTER
                      (*      INTERARRAY    :  AN ARBITRARRY ARRAY
                      (*      BA            :  BRANCH ADDRESS
                      (*      BAD           :    ,,     ,,     IN DECIMEL
                      (*      MAPO          :MAP.PROM OUTPUT ADDRESS
                      (*      MODA          :    ,,     ,,      ,,     IN DECIMEL
                      (*      VECA          :  VECTOR ADDRESS IN BINARRY
                      (*      VECD          :    ,,      ,,     ,, DECIMEL
                      (*      I             :  FOUR INSTRUCTION BITS IN Am 2910
                      (*      INS           :    ,,     ,,       EXPRESSED IN DECIMEL
                      (*      POINTER       :  POINTER TO THE STACK
                      (*      CCEN          :  CONDITION CODE ENABLE
                      (*      CC            :  CONDITION CODE INPUT
                      (*      RLD           :  REGISTER LOAD
                      (*      CI            :  CARRY-IN TO INCREMENTER
                      (*      PLE           :  PIPELINE ADDRESS ENABLE
                      (*      MAPE          :  MAP ADDRESS ENABLE
                      (*      VECTE         :  VCTOR ADDRESS ENABLE
                      (*      Y             :  MULIPLEXER OUTPUT
                      (*      YOUT          :  MICROPROGRAME ADDRESS
                      (*      OE            :  OUTPUT ENABLE
                      (*      MPCOUT        :  MICROPROGRAM COUNTER OUTPUT
                      (*      MUXIN1        :  MULTIPLEXER INPUT1
                      (*      MUXIN2        :     ,,          ,, 2
                      (*      MUXIN3        :     ,,          ,, 3
                      (*      MUXIN4        :     ,,          ,, 4
                      (*      DIN           :  DIRECT INPUTS
                      (*      REGOUT        :  REGISTER OUTPUT
                      (*      COUNTEROUT    :  COUNTER OUTPUT
                      (*      POPPEDNUM     :  POPPED NUMBER FROM THE STACK
                      (*      PUSHEDNUM     :  PUSHED   ,,     ,,   ,,  ,,
                      (*      MPCIN         :  MICROPROGRAME COUNTER INPUT
                      (*      MPCOUT        :      ,,            ,,     OUTPUT
                      (*      PL            :  PIPELINE OUTPUT
                      (*      -1            :  DENOTES TRI-STATE
                      (*      P,J,K,L,M,N   :  LOCAL VARIABLE
                      (*      SUM           :  LOCAL VARIABLE
                      (*
                      (* -----------------------------------------------------------
                 CONST
                      STACKSIZE=5;
```

```
II Pascal-86, V2.0


E NESTING        SOURCE TEXT: :F3:SEQ.SIM
  0  0                MAXCOUNT=4096;
  0  0           TYPE
                      STACKARRAY=array[1..stacksize]of integer;
  0  0                POINTERLIMITS=0..STACKSIZE;
  0  0                INTERARRAY=array[1..16]of integer;
  0  0           var
                    PL:array[1..64]of integer;
  0  0              BA,MAPO,VECA,I:INTERARRAY;
  0  0              STACK:STACKARRAY;
  0  0              POINTER:POINTERLIMITS;
  0  0              STACKEMPTY,STACKFULL:boolean;
  0  0              J,K,L,M,N,CCEN,RLD,CC,CI,MODA,VECD,BAD,INS,PLE,MAPE,VECTE,Y,MPCOU
                    MUXIN1,MUXIN2,MUXIN3,MUXIN4,DIN,REGOUT,POPPEDNUM,PUSHEDNUM:intege
  0  0              OE,YOUT:integer;
  0  0           INP:TEXT;
  0  0           OUT:TEXT;
  0  0           procedure BINTODEC(I:INTERARRAY;N:integer;var SUM:integer);
  1  0            var
                     J,P:integer;
  1  0            begin
  1  1                 J:=1;SUM:=0;P:=1;
  1  1                 repeat
  1  2                       if J=1 then SUM:=SUM+I[J]
  1  2                       else
  1  2                           begin
  1  3                                I[J]:=I[J]*2*P;
  1  3                                P:=2*P;
  1  3                                SUM:=SUM+I[J]
                                  end;
  1  2                       J:=J+1;
  1  2                       until J=N+1;
  1  1             end;{BINARY TO DEC CONVERSION}
  0  0           procedure   CLRSTACK(var POINTER:POINTERLIMITS;var STACK:STACKARRAY)
  1  0              begin
  1  1                   STACK[POINTER]:=0;
  1  1                   POINTER:=0
                    end;{CLEAR STACK.}
  0  0            procedure READSTACK(var POPPEDNUM:integer;var POINTER:POINTERLIMITS
  1  0                                 var STACK:STACKARRAY);
  1  0              begin
  1  1                   if POINTER>=1 then POPPEDNUM:=STACK[POINTER]
  1  1              end;{READ STACK.}
  0  0            procedure POP(var POPPEDNUM:integer;var POINTER:POINTERLIMITS;
  1  0                            var STACK:STACKARRAY;var STACKFULL,STACKEMPTY:boolean
  1  0              begin
  1  1                   if POINTER>0 then
  1  1                    begin
  1  2                         POPPEDNUM:=STACK[POINTER];
  1  2                         POINTER:=POINTER-1
                           end;
  1  1                     if POINTER=0 then
  1  1                       begin
  1  2                            STACKEMPTY:=true.
  1  2                            WRITELN('WARNING
                             end
```

```
I Pascal-86, V2.0


NESTING        SOURCE TEXT: :F3:SEQ.SIM
 1  1                      else STACKEMPTY:=false
 1  1              end; {OF POP.}
 0  0            procedure PUSH(PUSHEDNUM:integer; var POINTER:POINTERLIMITS;
 1  0                           var STACK:STACKARRAY; var STACKFULL,STACKEMPTY:boolear
 1  0              begin
 1  1                    if POINTER<STACKSIZE then
 1  1                    begin
 1  2                        STACKEMPTY:=false;
 1  2                        POINTER:=POINTER+1;
 1  2                        STACK[POINTER]:=PUSHEDNUM
                         end;
 1  1                    if POINTER=STACKSIZE then
 1  1                    begin
 1  2                        STACKFULL:=true;
 1  2                        WRITELN(OUT,'  WARNING:STACK FULL+CANNOT PUSH.')
                         end
 1  1                    else STACKFULL:=false
 1  1                    end;{OF PUSH}
 0  0            procedure LOADREG(var REGOUT:integer);
 1  0               begin
 1  1                    if (RLD=0) then
 1  1                    REGOUT:=DIN;
 1  1               end; {OF LOAD REG.}
 0  0           procedure CMPC(MPCIN:integer; var MPCOUT:integer);
 1  0               begin
 1  1                    if (CI=1)then MPCOUT:=MPCIN+1
 1  1                    else MPCOUT:=MPCIN;
 1  1               end; {OF CMPC}
 0  0            procedure DECREMENT(var COUNTEROUT:integer);
 1  0               begin
 1  1                    if RLD=0 then LOADREG(REGOUT)
 1  1                    else
 1  1                    begin
 1  2                        if REGOUT=0 then
 1  2                        WRITELN(OUT,'  COUNTER OUTPUT=0--CANNOT DECREMENT.')
                         else
 1  2                    COUNTEROUT:=(REGOUT-1)MOD MAXCOUNT;
 1  2                    end
 1  1               end; {OF DECREMENT.}
 0  0               begin{MAIN}
 0  1                    RESET(INP,':F2:SEQ.DAT');
 0  1                    REWRITE(OUT,':F2:SEQR.OUT');

 0  1                    repeat
 0  2                    for J:=1 to 32 do READ(INP,PL[J]);
 0  2                    READLN(INP);
 0  2                    for J:=33 to 64 do READ(INP,PL[J]);
 0  2                    READLN(INP);
 0  2                    CCEN:=PL[42];
 0  2                    RLD:=PL[41];
 0  2                    CI:=PL[43];
 0  2                    for K:=1 to 12 do
 0  2                    BA[K]:=PL[47+K];
 0  2                    for L:=1 to 12 do
 0  2                    READ(INP,MAPO[L]);
```

```
  2                 READLN(INP);
  2                 for M:=1 to 4 do
  2                 I[M]:=PL[43+M];
  2                 for N:=1 to 12 do
  2                 READ(INP,VECA[N]);
  2                 READLN(INP);
  2                 READLN(INP,CC);
  2                 N:=12;
  2                 BINTODEC(MAPO,N,MODA);
  2                 BINTODEC(VECA,N,VECD);
  2                 BINTODEC(BA,N,BAD);
  2                 N:=4;
  2                 BINTODEC(I,N,INS);

  2                 case INS of
  3                    0:
                         begin
  4                          WRITELN('JZ');
  4                          PLE:=0; MAPE:=1; VECTE:=1;
  4                          DIN:=BAD;
  4                          LOADREG(REGOUT);
  4                          Y:=0;
  4                          CLRSTACK(POINTER,STACK);
  4                          CMPC(Y,MPCOUT);
  4                          MUXIN4:=MPCOUT
                         end; {JZ}
  3                    1:begin
  4                          WRITELN('CJS');
  4                          PLE:=0;MAPE:=1;VECTE:=1;
  4                          DIN:=BAD;
  4                          LOADREG(REGOUT);
  4                          if (CCEN=1)or (CC=0) then
  4                          begin
  5                              DIN:=BAD;
  5                              MUXIN1:=DIN;
  5                              Y:=MUXIN1;
  5                              PUSH(MPCOUT,POINTER,STACK,STACKFULL,STACKEMPT
  5                              CMPC(Y,MPCOUT)
                                 end
  4                              else
  4                                 begin
  5                                     MUXIN4:=MPCOUT;
  5                                     Y:=MUXIN4;
  5                                     CMPC(Y,MPCOUT)
                                    end;
  4                          end; {CJS}
  3                    2:
                         begin
  4                          WRITELN('JMAP');
  4                          PLE:=1; MAPE:=0; VECTE:=1;
  4                          DIN:=MODA;
  4                          LOADREG(REGOUT);
  4                          MUXIN1:=DIN;
  4                          Y:=MUXIN1;
  4                          CMPC(Y,MPCOUT)
```

```
NESTING        SOURCE TEXT: :F3:SEQ.SIM
                             end{JMAP};
 0  3                    3:
                           begin
 0  4                           WRITELN('CJP');
 0  4                           PLE:=0;  MAPE:=1;
 0  4                           VECTE:=1;
 0  4                           DIN:=BAD;
 0  4                           LOADREG(REGOUT);
 0  4                           if (CCEN=1) or (CC=0) then
 0  4                           begin
 0  5                                MUXIN1:=DIN;
 0  5                                Y:=MUXIN1;
 0  5                                CMPC(Y,MPCOUT)
                                end
 0  4                           else
 0  4                           begin
 0  5                           MUXIN4:=MPCOUT;
 0  5                           Y:=MUXIN4;
 0  5                           CMPC(Y,MPCOUT)
                                end
 0  4                      end{CJP};
 0  3                4:
                       begin
 0  4                       WRITELN('PUSH');
 0  4                       PLE:=0;  MAPE:=1;VECTE:=1;
 0  4                       DIN:=BAD;
 0  4                       LOADREG(REGOUT);
 0  4                       PUSH(MPCOUT,POINTER,STACK,STACKFULL,STACKEMPTY);
 0  4                       MUXIN4:=MPCOUT;
 0  4                       Y:=MUXIN4;
 0  4                       CMPC(Y,MPCOUT);
 0  4                       if (CCEN=1) or (CC=0) then
 0  4                       begin
 0  5                           DIN:=BAD;
 0  5                           REGOUT:=DIN
                            end;
 0  4                  end{PUSH};
 0  3                5:
                       begin
 0  4                       WRITELN('JSRP');
 0  4                       PLE:=0;  MAPE:=1;  VECTE:=1;
 0  4                       DIN:=BAD;
 0  4                       LOADREG(REGOUT);
 0  4                       PUSH(MPCOUT,POINTER,STACK,STACKFULL,STACKEMPTY);
 0  4                       if (CCEN=1) or (CC=0) then
 0  4                       begin
 0  5                           DIN:=BAD;
 0  5                           MUXIN1:=DIN;
 0  5                           Y:=MUXIN1;
 0  5                           CMPC(Y,MPCOUT)
                            end
 0  4                       else
 0  4                       begin
 0  5                           MUXIN2:=REGOUT;
 0  5                           Y:=MUXIN2;
```

```
NESTING        SOURCE TEXT: :F3:SEQ.SIM
 0  5                          CMPC(Y,MPCOUT)
                          end;
 0  4                 end{JSRP};
 0  3            6:
                      begin
 0  4                      WRITELN('CJV');
 0  4                      PLE:=1;MAPE:=1;VECTE:=0;
 0  4                      DIN:=VECD;
 0  4                      LOADREG(REGOUT);
 0  4                      if (CCEN=1) or (CC=0)then
 0  4                      begin
 0  5                           DIN:=VECD;
 0  5                           MUXIN1:=DIN;
 0  5                           Y:=MUXIN1;
 0  5                           CMPC(Y,MPCOUT)
                           end
 0  4                      else
 0  4                      begin
 0  5                           MUXIN4:=MPCOUT;
 0  5                           Y:=MUXIN4;
 0  5                           CMPC(Y,MPCOUT)
                           end;
 0  4                 end{CJV};
 0  3            7:
                      begin
 0  4                      WRITELN('JRP');
 0  4                      PLE:=0; MAPE:=1; VECTE:=1;
 0  4                      DIN:=BAD;
 0  4                      LOADREG(REGOUT);
 0  4                      if (CCEN=1) or (CC=0) then
 0  4                      begin
 0  5                           DIN:=BAD;
 0  5                           MUXIN1:=DIN;
 0  5                           Y:=MUXIN1;
 0  5                           CMPC(Y,MPCOUT)
                           end
 0  4                      else
 0  4                      begin
 0  5                           MUXIN2:=REGOUT;
 0  5                           Y:=MUXIN2;
 0  5                           CMPC(Y,MPCOUT)
                          end;
 0  4                  end{JRP};
 0  3             8:
                       begin
 0  4                  WRITELN('RFCT');
 0  4                  PLE:=0; MAPE:=1; VECTE:=1;
 0  4                  DIN:=BAD;
 0  4                  LOADREG(REGOUT);
 0  4                  if (REGOUT=0) then
 0  4                  begin
 0  5                       MUXIN4:=MPCOUT;
 0  5                       Y:=MUXIN4;
 0  5                       POP(POPPEDNUM,POINTER,STACK,STACKFULL,STACKEMPTY);
 0  5                       CMPC(Y,MPCOUT)
```

```
ESTING       SOURCE TEXT: :F3:SEQ.SIM
0  4                  LOADREG(REGOUT);
0  4                  MUXIN4:=MPCOUT;
0  4                  Y:=MUXIN4;
0  4                  CMPC(Y,MPCOUT)
                 end{CONT};
0  3           15:
                 begin
0  4                  WRITELN('TWB');
0  4                  PLE:=0;  MAPE:=1;  VECTE:=1;
0  4                  DIN:=BAD;
0  4                  LOADREG(REGOUT);
0  4                  if( REGOUT=0) then
0  4                  begin
0  5                       if (CCEN=1) or (CC=0) then
)  5                       begin
)  6                            MUXIN4:=MPCOUT;
)  6                            Y:=MUXIN4;
)  6                            POP(POPPEDNUM,POINTER,STACK,STACKFULL,STACKEMPTY);
)  6                            MUXIN3:=POPPEDNUM;
)  6                            CMPC(Y,MPCOUT)
                           end
)  5                       else
)  5                       begin
)  6                            DIN:=BAD;
)  6                            MUXIN1:=DIN;
)  6                            Y:=MUXIN1;
)  6                            POP(POPPEDNUM,POINTER,STACK,STACKFULL,STACKEMPTY);
)  6                            MUXIN3:=POPPEDNUM;
)  6                            CMPC(Y,MPCOUT)
                           end;
)  5                  end
   4                  else
   4                  begin
   5                       if (CCEN=1) or (CC=0) then
   5                       begin
   6                            MUXIN4:=MPCOUT;
   6                            Y:=MUXIN4;
   6                            POP(POPPEDNUM,POINTER,STACK,STACKFULL,STACKEMPTY);
   6                            MUXIN3:=POPPEDNUM;
   6                            CMPC(Y,MPCOUT);
   6                            DECREMENT(REGOUT)
                           end
   5                       else
   5                       begin
   6                            READSTACK(POPPEDNUM,POINTER,STACK);
   6                            MUXIN3:=POPPEDNUM;
   6                            Y:=MUXIN3;
   6                            CMPC(Y,MPCOUT);
   6                            DECREMENT(REGOUT)
                           end;
   5                  end;
   4             end{TWB};
   3           end{OF CASE};
   2           if (OE=0) then YOUT:=Y else YOUT:=-1;
   2           WRITELN(OUT,'PLE=',PLE:4,'   MAPE=',MAPE:4,'   VECTE=',VECTE:4);
```

```
ESTING         SOURCE TEXT: :F3:SEQ.SIM
0  2             WRITELN(OUT,'BAD=',BAD:8,'   MODA=',MODA:8,'   VECD=',VECD:8,'DIN=',DIN
0  2             WRITELN(OUT,'MUXIN1=',MUXIN1:8,'   MUXIN2=',MUXIN2:8,'   MUXIN3=',MUXIN
               '   MUXIN4=',MUXIN4:8);
0  2             WRITELN(OUT,'Y=',Y:8,'   MPCOUT=',MPCOUT:8,'   POPPEDNUM=',POPPEDNUM:8,
                 '   REGOUT=',REGOUT:8);
0  2             WRITELN(OUT,'OE=',OE:4);
0  2             WRITELN(OUT,'YOUT=',YOUT:8);
0  2             WRITELN(OUT,'CCEN=',CCEN:4,'   CC=',CC:4,'   INS=',INS:4,'   CI=',CI:4,
                 '   RLD=',RLD:4);
0  2             WRITELN(OUT,'POINTER=',POINTER:4);
0  2             until EOF(INP);
0  1             end.



ormation:

                OFFSET    CODE SIZE      DATA SIZE     STACK SIZE
                015AH  0072H   114D                    0008H      8D
                01CCH  0026H    38D                    0006H      6D
                01F2H  0029H    41D                    0006H      6D
                021BH  0067H   103D                    000EH     14D
                0282H  006BH   107D                    000EH     14D
                02EDH  001AH    26D                    0006H      6D
                0307H  0026H    38D                    0006H      6D
                032DH  005AH    90D                    000EH     14D
                0387H  0E75H  3701D  0175H    373D  0028H     40D
IDE-                   015AH   346D

                       11FCH  4604D  0175H    373D  00A6H    166D

lead.
 Detected.
ation of Memory.
```